



Universiteit
Leiden

Entropy-based Lumping of Categorical Covariates for Regression Models

D.O. Koning (s3956105)

Supervisors:
Valentina Masarotto & Arno Knobbe

BSC THESIS— COMPUTER SCIENCE & MATHEMATICS

Leiden Institute of Advanced Computer Science (LIACS)
`liacs.leidenuniv.nl`

June 30, 2026

Abstract

Sparse categorical data is widely understood to cause a large array of problems. Levels having few data points exhibit larger random fluctuations than more populated levels, potentially leading to overfitting. A dimensionality reduction technique based on the combining of levels is studied. The criterion for what levels to lump together is mutual information, which has many desirable properties, such as being agnostic to the nature of the underlying relationship. Multiple different types of constraints, inherited from the type of data, are imposed on the combinations, ensuring a high degree of interpretability. This approach to dimensionality reduction turns out to be, in the general case, NP-complete, which is used to inform the design of the R package implementing the technique. The package is numerically demonstrated to improve the fit of a regression model in a small-sample setting.

Contents

Abstract	1
1 Introduction	3
2 Theoretical Background	5
2.1 Lumping	5
2.2 Information Theory	5
2.3 Choice of Estimator	8
2.3.1 Entropy Estimation	8
2.3.2 Mutual Information Estimation	9
2.4 Continuous Information Theory	10
3 The Unsupervised Problem	12
3.1 Problem Formalisation	12
3.2 Ordinal Case	12
3.3 Nominal Case	15
3.3.1 Characterisation	15
3.3.2 Algorithms	16
4 The Supervised Problem	22
4.1 Outcome-based Lumping	22
4.1.1 Discrete Outcomes	22
4.1.2 Continuous Outcomes	24
4.2 Context	27
5 Numerical Evaluation	29
5.1 Benchmark	29
5.2 Stability	29
5.3 Quality of Heuristics	31
5.4 Model Improvement	31
6 Conclusion	35
References	36
A Code	38

1 Introduction

Small sample sets often lead to problems, especially in the presence of a large number of covariates. For regression models, the more covariates are included, the more ability the model has to memorise the noise of the sample it is trained on, rather than fitting the actual underlying relationship. This overfitting leads to poor predictive power outside of the training data and bad inference [Har01, Ch. 4].

A large number of methods have been proposed to remedy this problem [HTF09]. In this thesis we investigate the method of lumping, which involves combining multiple levels of the same discrete covariate together to reduce the dimensionality of the problem. This technique already has some existing backing¹, but we will extend it by basing it on information theory, in particular mutual information. We will choose that way of lumping the variables that is the most informative.

Mutual information has a number of advantages when it comes to using it as a metric. For one, it is able to detect any type of relationship, whereas other metrics make certain assumptions. A R^2 score will only ever be able to detect linear relationships and might entirely neglect sinusoidal ones. This means mutual information uniquely is agnostic as to the regression model that will be fit after the dimensionality reduction is carried out, unlike other metrics [Ros14]. Furthermore, the mutual information has an easy interpretation, both in terms of its nature—it is just the strength of the relationship between two variables—and its value. It carries units of bits, which have an intuitive meaning, aiding explainability.²

The usefulness of information theory for the present application is widely recognised, being used for subgroup discovery [KH06] and the discretisation of continuous variables [DKS95]. Most similarly to our algorithm is COOLCAT, which assigns observations to a category based on its informational content [BLC02]. Our approach to lumping is distinct, in that it does not make a new compound category composed of multiple variables. Instead, it simply combines multiple levels of the same variable together. This simplicity allows our solver to be exact, rather than simply a greedy heuristic, and also allows the imposition of restrictions on which combinations are allowed.

The requirements for which combinations we allow depend on the type of variable being considered. Ordinal variables only admit the combination of adjacent levels, increasing the interpretability of the resulting levels, since this prevents the creation of strange levels. For nominal variables, we will take as an input what levels may be combined. For the software implementation, this means the user is provided the additional flexibility to use their domain expertise to tweak the behaviour of the algorithms.

Beyond this distinction of variable types, we also study two different types of information to optimise for. In line with broader machine learning terminology [HTF09], we distinguish between unsupervised and supervised lumping. In unsupervised lumping, only the distribution of the variable being lumped is taken into account, making for a simpler problem. By contrast, supervised lumping uses the joint distribution of the variable being lumped and the outcome, meaning

¹See e.g. Wickham [Wic25] and Harrell [Har01].

²See remarks 2.5 and 2.15.

it has the ability to use more information. This means supervised lumping has a potentially better fit to the data, at the cost of being more complicated and carrying the risk of overfitting the lumping to the training data.

An R package is developed to implement the algorithms and techniques discussed in this thesis, for which, see appendix A.

We begin by stating some relevant background information, particularly from information theory in section 2. Then, in section 3, we will derive some facts and state the algorithms for unsupervised lumping, an approach which section 4 mirrors for supervised lumping. Finally, section 5 will evaluate the performance of lumping on some synthetic datasets.

2 Theoretical Background

2.1 Lumping

Thus far we have spoken informally of the notion of lumping. It is useful to state precisely what lumping means.

Definition 2.1. A *lumping* $L' = \{l'_1, \dots, l'_{m'}\}$ of a categorical variable with levels $L = \{l_1, \dots, l_m\}$ is a partition of L . The elements of L' are referred to as *lumps* or *lumped levels*.³

Remark 2.2. Throughout this thesis, we will, by abuse of notation, identify the level set L with its set of singletons. This corresponds to the trivial lumping which does not combine any levels together.

Remark 2.3. We will also use *to lump* as a verb, for the act of mapping L to L' and for the combining/*lumping* of two or more levels.

2.2 Information Theory

We begin by collecting some elementary results from information theory, largely following Cover and Thomas [CT91, Ch. 2]. Results taken directly from that work are stated without proof. All other statements, regardless of origin, are proven. We also adopt their notation, with the notable exception that we do not use the notation $p(x)$, for the probability of a random variable taking the value x , instead we will always explicitly write the random variable in question, $\mathbb{P}(X = x)$, in accordance with Grimmett and Welsh [GW14].

The first important notion is that of entropy. The entropy is a functional and measures how much uncertainty exists about a random variable. If a variable is highly disordered, its entropy will be high and when its distribution is concentrated in a small number of values, it is more predictable and its entropy is low.

Definition 2.4. The *entropy* of a discrete random variable X taking values in $\mathcal{X}$ is

$$H(X) := - \sum_{x \in \mathcal{X}} \mathbb{P}(X = x) \log \mathbb{P}(X = x),$$

where—as throughout this thesis—we adopt the convention that $0 \log 0 = 0$.

Remark 2.5. All logarithms in this text are natural. This is a matter of convention and does not convey mathematical meaning, since a change of base only results in definition 2.4 carrying different units—bits in the case of base 2, nats for base e , and dits for base 10. For the later optimisation problem, the units do not matter, since they can be converted into one another by a constant multiplicative factor [CT91, Lem. 2.1.2.].

A logical next question is to try to generalise this notion to be able to condition variables. This is possible and gives an important quantity, but it is not done in the immediately obvious manner—by replacing all the probabilities with conditional probabilities—instead, the following definition is chosen for its convenient properties.

³This vocabulary—but not the definition—is inspired by Wickham [Wic25].

Definition 2.6. Let X and Y be discrete random variables taking values in $\mathcal{X}$ and $\mathcal{Y}$, respectively. The *conditional entropy* of Y given X is

$$H(Y|X) := - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} \mathbb{P}(X = x, Y = y) \log \mathbb{P}(Y = y|X = x).$$

The conditional entropy can be thought of as the amount of entropy or uncertainty remaining as to the distribution of Y once X has been realised. It is important to distinguish this from the next important concept: mutual information, which quantifies how much one learns about the value of one variable upon observing the value of another.

Definition 2.7. The *mutual information* between two discrete random variables X and Y taking values in $\mathcal{X}$ and $\mathcal{Y}$, respectively, is

$$I(X; Y) := \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} \mathbb{P}(X = x, Y = y) \log \frac{\mathbb{P}(X = x, Y = y)}{\mathbb{P}(X = x)\mathbb{P}(Y = y)}.$$

Remark 2.8. An equivalent definition of mutual information exists, which lies beyond the scope of this thesis, but may help with intuition. The mutual information is also the Kullback–Leibler divergence between the joint distribution and the product of the marginal distributions.

It is clear that the concepts of conditional entropy and mutual information are closely related. In a sense, they are each other’s dual: mutual information says how much uncertainty about one variable is removed by observing another and the conditional entropy is how much uncertainty remains. This interpretation is formalised by the following lemma.

Lemma 2.9. For discrete random variables X and Y ,

$$I(X; Y) = H(Y) - H(Y|X).$$

A similar decomposition exists, which formalises the same intuition as lemma 2.9 in a different manner. We later use this to estimate the mutual information.

Lemma 2.10. For discrete random variables X and Y ,

$$I(X; Y) = H(X) + H(Y) - H(X, Y).$$

The symmetry in definition 2.7 immediately implies lemma 2.11, but it is included here for completeness. This result will be used implicitly throughout the thesis.

Lemma 2.11. For discrete random variables X and Y ,

$$I(X; Y) = I(Y; X).$$

Lemma 2.12. For independent discrete random variables X and Y ,

$$I(X; Y) = 0.$$

Proof. By definition, variables are independent when their joint distribution is the product of their marginal distributions, so

$$\log \frac{\mathbb{P}(X = x, Y = y)}{\mathbb{P}(X = x)\mathbb{P}(Y = y)} = \log 1 = 0.$$

Hence, the mutual information becomes the expectation of zero, which is zero. $\square$

Lemma 2.13. *For all discrete random variables X taking values in $\mathcal{X}$ and functions $f : \mathcal{X} \rightarrow \mathcal{Y}$,*

$$H(f(X)) \leq H(X).$$

with equality if and only if f is invertible on the support of X .

Lemma 2.13 captures the intuition that transforming a variable can only reduce the information available. In other words, no matter how a variable is manipulated, it is not possible to extract more information than it originally contains.

Theorem 2.14 ([Gra11, Lem. 2.5.2]). *For any random variable X taking values in a finite set $\mathcal{X}$ and any function $f : \mathcal{X} \rightarrow \mathcal{Y}$, we have*

$$I(X; f(X)) = H(f(X)).$$

This theorem says that observing the transformed version of a variable can only tell you as much about the original variable as the entropy of the transformed version. The key idea of the proof is that, since functions are deterministic, if the value of X is known, no uncertainty can remain about the value of $f(X)$, so $H(f(X)|X) = 0$.

Proof of theorem 2.14. Expanding the conditional entropy, we obtain

$$H(f(X)|X) = - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} \mathbb{P}(X = x, f(X) = y) \log \mathbb{P}(f(X) = y|X = x). \quad (2.1)$$

Notice that $f(X) = y$ occurs if and only if $f(x) = y$, meaning

$$\mathbb{P}(X = x, f(X) = y) = \begin{cases} \mathbb{P}(X = x) & f(x) = y, \\ 0 & \text{otherwise} \end{cases}.$$

All terms of the sum with $f(x) \neq y$ are thus zero, allowing us to simplify equation (2.1) by removing a sum and replacing each occurrence of y with $f(x)$:

$$\begin{aligned} H(f(X)|X) &= - \sum_{x \in \mathcal{X}} \mathbb{P}(X = x, f(X) = f(x)) \log \mathbb{P}(f(X) = f(x)|X = x) \\ &= - \sum_{x \in \mathcal{X}} \mathbb{P}(X = x) \log 1 \\ &= 0, \end{aligned}$$

which, when substituted in lemma 2.9, yields the result. $\square$

Remark 2.15. Combining lemma 2.12 and theorem 2.14, gives another way of interpreting the mutual information: it measures the *degree of dependence* between X and Y . When they are as dependent as possible—which happens when one is a function of the other—theorem 2.14 says they contain as much information as the least informative of the two⁴ and when they are independent, they contain no information about each other. In between those two extremes, the ‘more independent’ they are, the closer $\mathbb{P}(X = x, Y = y)$ becomes to $\mathbb{P}(X = x)\mathbb{P}(Y = y)$, so the lower the mutual information between them.

Theorem 2.14 is useful in several ways. It establishes an equality between the mutual information and entropy for a restricted class of variables, enabling the application of results about mutual information and entropy at the same time. It also allows us to estimate mutual information by estimating entropy instead, which is, in general, easier (see section 2.3.2). This is especially important since the lumping process is an example of such a function,⁵ meaning an estimator for $H(f(X))$ (or thus equivalently $I(X; f(X))$) ends up being the way we evaluate the success of a lumping.

2.3 Choice of Estimator

With real data, it is not possible to work with the actual entropy or mutual information, since they rely on the probability distribution generating the data, which is a priori unknown. There is thus a need to estimate them. This subsection outlines a number of estimators and explains the choice made.

2.3.1 Entropy Estimation

The most obvious way to estimate entropy is to replace all the probabilities in definition 2.4 with the empirical probabilities. In other words, to simply ‘plug-in’ an estimate of the probabilities. This turns out to yield a very good estimator:

Definition 2.16 ([AK01, p. 5]). Let $X \in \mathcal{X}^n$. The *plug-in estimator* or *empirical entropy* for the entropy of a random variable taking values in a finite set $\mathcal{X}$ is

$$\hat{H}_{\text{emp}}^{\mathcal{X}}(X) := - \sum_{x \in \mathcal{X}} \hat{p}_x \log \hat{p}_x,$$

where $\hat{p}_x = \frac{1}{n} \sum_{i=1}^n \mathbb{1}(X_i = x)$ are the empirical probabilities.

Remark 2.17. Where it is clear from context, we do not write the sample explicitly and use $\hat{H}_{\text{emp}}^{\mathcal{X}}$ to denote $\hat{H}_{\text{emp}}^{\mathcal{X}}(X)$.

The empirical entropy enjoys the advantage of being the Maximum Likelihood Estimator for the entropy [Pan03, p. 1197] as well as strong universal consistency [AK01, Cor. 1]. It does suffer from significant downwards bias, as demonstrated by lemma 2.18.

Lemma 2.18 ([Pan03, Ineq. 3.5]). *For all random variables X taking values in a finite set $\mathcal{X}$,*

$$\mathbb{E}(\hat{H}_{\text{emp}}^{\mathcal{X}}) \leq H(X)$$

⁴This is a consequence of lemma 2.13, since the least informative one must be the transformed version of the other.

⁵In particular, set $\mathcal{Y} = \mathcal{P}(\mathcal{X})$, the powerset of $\mathcal{X}$, and take f to be the function that sends a level to the lumped level it ends up in.

Proof. Notice $p \mapsto -p \log p$ is strictly concave. Jensen's inequality [Jen06, Ineq. 4] then implies that, for each $x \in \mathcal{X}$,

$$\mathbb{E}(-\hat{p}_x \log \hat{p}_x) \leq -\mathbb{E}(\hat{p}_x) \log \mathbb{E}(\hat{p}_x) = -\mathbb{P}(X = x) \log \mathbb{P}(X = x),$$

where the last equality follows from the unbiasedness of the empirical probabilities. Summing over all $x \in \mathcal{X}$ then yields the lemma. $\square$

To circumvent this, the Miller-Madow bias-corrected estimator has been proposed, which tries to adjust the empirical entropy upwards in order to compensate the bias:

Definition 2.19 ([Pan03, p. 1197]). Let $X \in \mathcal{X}^n$. The *Miller-Madow bias-corrected estimator* for the entropy of a random variable taking values in a finite set $\mathcal{X}$ is

$$\hat{H}_{\text{MM}}^{\mathcal{X}}(X) := \hat{H}_{\text{emp}}^{\mathcal{X}}(X) + \frac{\hat{m} - 1}{2n},$$

where $\hat{m}$ is the number of $x \in \mathcal{X}$ having $\hat{p}_x > 0$.

We note here that the Miller-Madow estimator is not a good choice of estimator for the present application, since maximising for the Miller-Madow estimator is equivalent to maximising the empirical entropy, as either way, all of the levels with zero samples need to be lumped into another category. This means that the maximum of the Miller-Madow estimator is the same as the maximum of the empirical entropy, up to the additive constant $-\frac{1}{2n}$. Both goal functions will thus reward the same lumpings the same way, modulo the categories the levels with zero samples get assigned to, which is acceptable, since those are arbitrary regardless.

A variety of other, more exotic estimators have also been developed in order to correct the bias, such as the Grassberger estimator [KSG04, Eq. 35], the James-Stein shrinkage estimator [HS09] and the jackknife [Pan03, p. 1198]. However, Paninski [Pan03, Prop. 8] shows that all estimators for entropy are biased, meaning the best we can aim for is a reduction in bias, not the complete elimination.

Given this restriction, the most sensible estimator is the empirical entropy, since it limits the advantage other estimates are able to have over it, meaning that the fact it is the Maximum Likelihood Estimator becomes an overwhelming factor. On top of that theoretical benefit, there is significant practical merit to the simplicity of the empirical entropy. As we will show later,⁶ the sum structure it has means the entropy lost by lumping becomes an easy-to-calculate expression, simplifying optimisation greatly.

2.3.2 Mutual Information Estimation

The obvious estimator for mutual information is obtained by analogy to definition 2.16 and is motivated by lemma 2.10.

Lemma and definition 2.20 ([Pan03, p. 1204]). *The maximum likelihood estimator for the mutual information between two random variables X and Y*

⁶See e.g. sections 3.2 and 4.1.1.

taking values in finite sets $\mathcal{X}$ and $\mathcal{Y}$, respectively, is the plug-in estimator or empirical (mutual) information and is given by

$$\hat{I}_{emp}^{\mathcal{X},\mathcal{Y}}(X; Y) = \hat{H}_{emp}^{\mathcal{X}}(X) + \hat{H}_{emp}^{\mathcal{Y}}(Y) - \hat{H}_{emp}^{\mathcal{X},\mathcal{Y}}(X, Y).$$

Remark 2.21. Similarly to for definition 2.16, to avoid clutter, when it is clear from context the sample will not be written explicitly. Furthermore, we will usually only denote the level set of the variable being lumped in the superscript.

2.4 Continuous Information Theory

The results in section 2.2 are only stated for discrete variables. Real data often includes real-valued variables. Fortunately, information theory is rich enough to also treat continuous variables. We merely state informally that this happens in the way intuitively expected—probability mass functions turn into densities and sums turn into integrals. Every lemma or theorem stated in section 2.2 also applies to continuous variables. For a formal treatment see Cover and Thomas [CT91, Ch. 8]. In light of this, treating discrete and continuous separately is easy. More difficulty arises when mixing discrete and continuous variables.

Definition 2.22. The *mutual information* between two variables X and Y taking values in a finite $\mathcal{X}$ and $\mathcal{Y} \subseteq \mathbb{R}$, respectively, is

$$\begin{aligned} I(X; Y) &:= \sum_{x \in \mathcal{X}} \int_{\mathcal{Y}} p_{X,Y}(x, y) \log \frac{p_{X,Y}(x, y)}{p_X(x)p_Y(y)} dy \\ &= \int_{\mathcal{Y}} \sum_{x \in \mathcal{X}} p_{X,Y}(x, y) \log \frac{p_{X,Y}(x, y)}{p_X(x)p_Y(y)} dy, \end{aligned}$$

where

- p_X is the probability mass function of X ,⁷
- p_Y is the probability density function of Y , and
- $p_{X,Y}$ is the joint probability mass/density function of X and Y .

It is immediately apparent why our standard estimators no longer work in this instance; the probability of observing any one value is zero. Direct adaptations of definition 2.16 and lemma and definition 2.20 involve numeric integration. These are unsuitable for the present application, because the motivation behind lumping is that it should take away problems associated with small sample sizes, meaning standard integration techniques struggle with stability. For this reason, an alternative estimator for mutual information is used:

Definition 2.23 ([Ros14, Eq. 2]). Given finite set $\mathcal{X}$, a sample $(X_1, Y_1), \dots, (X_n, Y_n) \in \mathcal{X} \times \mathbb{R}$, and a parameter $k \in \mathbb{N}$, the *Ross estimator* for $I(X; Y)$ is

$$\hat{I}_{Ross}^{\mathcal{X}}(X; Y) := \psi(n) + \psi(k) - \frac{1}{n} \sum_{i=1}^n (\psi(n_i^{\mathcal{X}}) + \psi(c_i^{\mathcal{X}})),$$

where

⁷We deviate from our standard notation here, in order to be able to show the symmetry between the mass function of X and the density of y better.

- ψ is the digamma function [IK77, App. A, Table 17.I];
- $n_i^{\mathcal{X}} = \sum_{j=1}^n \mathbb{1}(X_i = X_j)$ is the number of data points having the same value of X as X_i ;
- $d_i^{\mathcal{X}}$ is the distance to the k^{th} closest point amongst the $n_i^{\mathcal{X}}$ data points having the same value of X as X_i ;
- $c_i^{\mathcal{X}} = \sum_{j=1}^n \mathbb{1}(|Y_i - Y_j| \leq d_i^{\mathcal{X}})$ is the number of data points being closer than $d_i^{\mathcal{X}}$ to Y_i .

Remark 2.24. For simplicity, we will only ever use $k = 3$. This is the value recommended by Ross [Ros14] and Kraskov, Stögbauer, and Grassberger [KSG04].

3 The Unsupervised Problem

3.1 Problem Formalisation

In order to be able to solve the problem it is necessary to formulate it as a mathematical optimisation problem. The considerations in section 2.3 motivate the choice of empirical entropy as an objective function. An objective alone does not suffice, because some constraints need to be imposed to restrict what lumpings are valid. Without such constraints, the optimal algorithm would always be to do nothing, as lemma 2.13 implies this is the lumping with maximal entropy. This is achieved by specifying a natural q as the minimum number of samples each lump must contain. Forcing the algorithm to deviate from the trivial lumping in this manner has the additional advantage of guaranteeing the elimination of excessively small levels.

Beyond these basic conditions, it is also desirable to impose further constraints on what lumps are admissible. For ordinal variables, it makes sense to require the lumped levels are contiguous, since this makes the interpretation substantially easier. For nominal variables, it is not possible to determine merely based on the level set what lumps should be allowed. In order to resolve this we need the notion of a preference set.

Definition 3.1. The *preference set* of a level $l \in L$ of a nominal variable with level set L is notated $P_l \subseteq L$ and contains at least l .

The interpretation of the preference set is simple; P_l is the set of levels that l is allowed to be lumped with. We will take the preference sets as an input, relying on domain expertise to determine what lumpings are sensible. For instance, if L is the set of countries, it makes sense to impose that the Netherlands can be lumped with Germany, but not with China. With these requirements, we can formulate the problem.

Problem 3.2. Given a categorical variable with levels L , a $q \in \mathbb{N}$, and $n \geq m$ samples $X_1, \dots, X_n \in L$, find a lumping $L' = \{L'_1, \dots, L'_m\}$ with maximal empirical entropy $\hat{H}_{\text{emp}}^{L'}(X)$ such that:

1. For each $1 \leq i \leq m'$, $D_i(L') \geq q$, where $D_i(L') := \sum_{j=1}^n \mathbb{1}(X_j \in l'_i)$ is the number of samples having level l'_i under lumping L' .
2. If L is ordinal, each element of L' is contiguous.
3. If L is nominal, for each lumped level $A \in L'$, it holds that if $a, b \in A$, then $a \in P_b$.

Remark 3.3. We will refer to constraint 3.2.1 as the *threshold constraint* and to constraints 3.2.2 and 3.2.3 as the *structure* or *preference constraints*.

3.2 Ordinal Case

To solve the ordinal problem, we evaluate the information loss induced by lumping levels $l_i \in L$ through levels $l_j \in L$ together to obtain the lumping L' . We note two facts about the distribution of this newly lumped variable:

- The probability of observing the new level is the sum of the probabilities of the levels l_i through l_j .

- The distribution of levels not between l_i and l_j is unaffected.

This means the estimate of entropy after the lumping becomes

$$\hat{H}_{\text{emp}}^{L'} = - \left(\sum_{k=i}^j \hat{p}_k \right) \log \left(\sum_{k=i}^j \hat{p}_k \right) - \sum_{\substack{1 \leq k < i \\ j < k \leq m}} \hat{p}_k \log \hat{p}_k. \quad (3.1)$$

The first term in equation (3.1) is the contribution to the entropy of the lumped level and the second term corresponds to the unaffected levels. Perhaps this is illustrated better by isolating the terms corresponding to levels l_i through l_j in definition 2.16 in the same manner:

$$\hat{H}_{\text{emp}}^L = - \sum_{k=i}^j \hat{p}_k \log \hat{p}_k - \sum_{\substack{1 \leq k < i \\ j < k \leq m}} \hat{p}_k \log \hat{p}_k.$$

With these facts, an explicit expression for the entropy lost in the transition from L to L' can be obtained:

$$J(i, j) := \hat{H}_{\text{emp}}^L - \hat{H}_{\text{emp}}^{L'} = \left(\sum_{k=i}^j \hat{p}_k \right) \log \left(\sum_{k=i}^j \hat{p}_k \right) - \sum_{k=i}^j \hat{p}_k \log \hat{p}_k. \quad (3.2)$$

Remark 3.4. This expression is only able to express how much information is lost from creating one contiguous lump. Notice, however, that $J(i, j)$ only depends on the number of samples having levels l_i through l_j , meaning it is unaffected by if any other non-overlapping lumping has been carried out, provided only levels smaller than l_i or greater than l_j are lumped. It thus generalises quite well, as the entropy loss of lumping several times can be calculated as the sum of the entropy loss of each of the individual disjoint contiguous ranges, irrespective of order.

Lemma 3.5. *If*

$$L' = \{\{l_1, \dots, l_{b_1}\}, \{l_{b_1+1}, \dots, l_{b_2}\}, \dots, \{l_{b_{m'}-1+1}, \dots, l_m\}\}$$

is an optimal lumping of $L = \{l_1, \dots, l_m\}$, then, for each b_k , the truncated version of L' ,

$$L' \mid_{b_k} := \{\{l_1, \dots, l_{b_1}\}, \{l_{b_1+1}, \dots, l_{b_2}\}, \dots, \{l_{b_{k-1}+1}, \dots, l_{b_k}\}\}$$

is also an optimal lumping of the prefix $L \mid_{b_k} := \{l_1, \dots, l_{b_k}\}$.

Proof. Clearly, all lumps in $L' \mid_{b_k}$ are contiguous and meet the threshold requirement, because L' meets the threshold requirement. It is thus a valid lumping.

To see that $L' \mid_{b_k}$ is also optimal, we proceed by contradiction. Suppose $L' \mid_{b_k}$ is not optimal, i.e. there exists another lumping of $L \mid_{b_k}$ with lower information loss. Write this lumping as $\{\{l_1, \dots, l_{c_1}\}, \dots, \{l_{c_{k-1}+1}, \dots, l_{b_k}\}\}$. Then

$$\{\{l_1, \dots, l_{c_1}\}, \dots, \{l_{c_{k-1}+1}, \dots, l_{b_k}\}, \{l_{b_k+1}, \dots, l_{b_{k+1}}\}, \dots, \{l_{b_{m'}-1+1}, \dots, l_m\}\}$$

is also a valid lumping of L , which—in light of remark 3.4—must have lower entropy loss than L' . $\nexists$ $\square$

Algorithm 3.1 Unsupervised Ordinal Lumping

```

1: procedure LUMPORDINAL( $\hat{\mathbf{p}}, q$ )
2:    $m \leftarrow$  length of  $\hat{\mathbf{p}}$ 
3:   Initialise  $f[0 \dots m]$  with  $\infty$ 
4:    $f[0] \leftarrow 0$ 
5:   Initialise  $CP[1 \dots m]$  with Null  $\triangleright$  Store cut points for backtracking
6:   for  $j \leftarrow 1$  to  $m$  do
7:     for  $i \leftarrow 1$  to  $j$  do
8:       if  $C(i, j) \geq q$  then
9:          $curr\_val \leftarrow f[i - 1] + J(i, j)$ 
10:        if  $curr\_val < f[j]$  then
11:           $f[j] \leftarrow curr\_val$ 
12:           $CP[j] \leftarrow i$ 
13:        end if
14:      end if
15:    end for
16:  end for
17:   $L' \leftarrow []$   $\triangleright$  Backtrack to construct final lumping
18:   $curr \leftarrow m$ 
19:  while  $curr > 0$  do
20:     $start\_i \leftarrow CP[curr]$ 
21:    Prepend  $start\_i$  to  $L'$ 
22:     $curr \leftarrow start\_i - 1$ 
23:  end while
24:  Append  $m + 1$  to  $L'$ 
25:  return  $L'$ 
26: end procedure

```

Remark 3.4 and lemma 3.5 together point to a solution based on these prefixes. We can iteratively build up a solution, taking into account one extra level of the variable at a time. The final entropy is then the sum of the individual losses.⁸ To see this define $f(j)$ to be the minimum entropy loss possible for a valid lumping of $\{l_1, \dots, l_j\}$. By setting $f(0) = 0$ we obtain the following dynamic program for f :

$$f(j) = \min_{\substack{1 \leq i \leq j \\ C(i, j) \geq q}} \{f(i - 1) + J(i, j)\}, \quad (3.3)$$

where we adopt the convention that $\min \emptyset = \infty$ and $C(i, j) = \sum_{k=i}^j D_k(L)$ the number of samples having levels l_i through l_j . Solving problem 3.2 is then equivalent to calculating $f(m)$, the straightforward method for which is given in algorithm 3.1.

Lemma 3.6. *Algorithm 3.1 has time complexity $O(m^3)$.*

Proof. Computing $J(i, j)$ requires sums of $j - i + 1$ terms. The worst case for this is $i = 1$ and $j = m$, so this computation is $O(m)$. The loop to calculate $f(m)$ runs j from 1 to m and for each value of j runs i from 1 to j . The amount

⁸Here we have technically also made use of the fact that $J(i, j)$ is non-negative due to lemma 2.13.

of loop iterations is thus $\sum_{j=1}^m j = O(m^2)$. For each loop iteration, $J(i, j)$ is computed, giving a total complexity of $O(m) \cdot O(m^2) = O(m^3)$. $\square$

3.3 Nominal Case

When looking at nominal variables, the loss of an ordering makes for a materially different problem. Most immediately, the number of possible lumpings explodes, but more subtly, the existence of the preference sets introduces the possibility of complicated structures in a way that does not exist for ordinal variables. Throughout this subsection, we will assume $P \neq NP$.

3.3.1 Characterisation

The first result we state about the nominal problem is a somewhat negative one about its computational complexity. In order to be able to do so we first restate problem 3.2 as a decision problem.

Problem 3.7. Given a categorical variable with levels L , a sample $X_1, \dots, X_n \in L$, a $q \in \mathbb{N}$ and a target entropy H , does there exist a lumping $L' = \{L'_1, \dots, L'_{m'}\}$ of L such that:

- For each $1 \leq i \leq m'$, $D_i(L') \geq q$.
- For each lumped level $A \in L'$, it holds that if $a, b \in A$, then $a \in P_b$.
- $\hat{H}_{\text{emp}}^{L'}(X_1, \dots, X_n) \geq H$.

It is clear that this is a strictly easier problem than problem 3.2, because, given a polynomial time algorithm for problem 3.2, one could immediately decide problem 3.7 as well. Even then, this problem is largely infeasible to solve directly.

Lemma 3.8. *Problem 3.7 is NP-complete.*

Proof. The problem is trivially NP. We thus only need to check if it is NP-hard. We proceed by a reduction from the partition problem: given a multiset A of positive integers, can A be partitioned into two subsets with equal sum [BCL92, p. 5]?

We map an instance of the partition problem to an instance of problem 3.7 as follows: For each element of A , have one corresponding element in L (we cannot have direct equality since A is a multiset and L is a regular set, so we must be careful to distinguish them; duplicate elements of A map to distinct elements of L so that $|L| = |A|$). Pick a sample such that $D_i(L) = a_i$ for all $i \in L$, which is clearly always possible. Set $q = \frac{1}{2} \sum A$. Finally, set $P_i = L$ for all $i \in L$ and choose target entropy $H = \log 2$.

Suppose we have a yes-instance of the partition problem, so there exist $A_1, A_2 \subset A$ with $A = A_1 \sqcup A_2$ and $\sum A_1 = \sum A_2$. Then we can choose the lumping $L' = \{L'_1, L'_2\}$, with L_i consisting of all the levels corresponding to A_i . This lumping satisfies all three constraints:

- We have

$$D_i(L) = \sum A_i = \frac{1}{2} \sum A = q.$$

- Since every preference set is all of L , they are clearly respected.

- To calculate the empirical entropy, first notice that the sample size is

$$n = \sum A = 2q,$$

so the entropy becomes

$$\begin{aligned}\hat{H}_{\text{emp}}^{L'}(S_1, \dots, S_n) &= -\frac{D_1(L')}{n} \log \left(\frac{D_1(L')}{n} \right) - \frac{D_2(L')}{n} \log \left(\frac{D_2(L')}{n} \right) \\ &= -\frac{q}{2q} \log \left(\frac{q}{2q} \right) - \frac{q}{2q} \log \left(\frac{q}{2q} \right) = \log 2.\end{aligned}$$

All constraints are satisfied, so this is also a yes-instance of problem 3.7.

Now suppose the instance of problem 3.7 is a yes-instance. Since the total sample size is $2q$, which is the sum of the number of samples having each level, and each level must have at least q samples, L' can have at most 2 levels. If $|L'| = 1$, then

$$\hat{H}_{\text{emp}}^{L'} = -\frac{n}{n} \log \left(\frac{n}{n} \right) = 0,$$

violating $\hat{H}_{\text{emp}}^{L'} \geq H = \log 2$, so L' must have more than one level. We conclude $L' = \{L'_1, L'_2\}$. Since $D_i(L') \geq q$ and $D_1(L') + D_2(L') = 2q$, we must have $D_1(L') = D_2(L') = q$. Finally, take A_i to be the subset of A corresponding to L'_i . A lumping is a partition of L , so A_1 and A_2 form a partition of A and

$$\sum A_i = D_1(L') = q = D_2(L') = \sum A_j,$$

showing that this is also a yes instance of the partition problem.

We therefore have a (clearly polynomial time) way of transforming an instance of the partition problem into an instance of problem 3.7, such that the original problem is a yes-instance if and only if the transformed problem is. Since we have a polynomial reduction from the partition problem, which is NP-hard [BCL92, Thm. 25], we conclude problem 3.7 is also NP-hard. $\square$

3.3.2 Algorithms

Whilst lemma 3.8 implies that for large m it is impossible to calculate exactly the optimal lumping, we will still start by providing an exact algorithm, both out of theoretical interest and the practical observation that m often is small for real-world datasets. Afterwards, several heuristics are developed for when there are enough levels to render precise computation infeasible.

Integer Linear Programming In order to devise an exact solution, we need two pieces of vocabulary.

Definition 3.9. The *preference graph* G_L of a nominal variable with level set L is the graph with vertices L and an edge between $a \in L$ and $b \in L$ if and only if $b \in P_a$ and $a \in P_b$.

Definition 3.10. A *clique* of a graph G is a complete subgraph of G .

By construction, two levels are only allowed to be lumped together if and only if there exists an edge between them in the preference graph. It follows that the cliques of the preference graph correspond directly to lumps that are allowed by the preference constraint, so we let $\mathcal{B} = \{B_1, \dots, B_N\}$ be the set of all cliques of the preference graph of a nominal variable L .

Since we are only interested in the lumpings satisfying the threshold constraint as well, $\mathcal{B}$ needs to be filtered. By discarding all cliques $B_i \in \mathcal{B}$ for which the number of samples having level B_i ,

$$N(B_i) := \sum_{l \in B_i} D_l(L) = n \sum_{l \in B_i} \hat{p}_l$$

is less than q we obtain $\mathcal{C} = \{C_1, \dots, C_K\} \subseteq \mathcal{B}$, the set of all valid cliques (and hence lumps).

Now inspect definition 2.16 more carefully. It is the sum of a simple term over all lumped levels. By assigning a weight

$$w_i = -\frac{N(C_i)}{n} \log \frac{N(C_i)}{n} \quad (3.4)$$

to each $C_i \in \mathcal{C}$, the empirical entropy is just the sum of the weights of all chosen lumps. By introducing binary variables $x_1, \dots, x_K$ that indicate if C_i is chosen in the final lumping, we find

$$\hat{H}_{\text{emp}}^{L'} = \sum_{i=1}^K x_i w_i.$$

By then requiring that each level is covered exactly once, we obtain the following integer linear program:

$$\begin{aligned} & \text{Maximise} && \sum_{i=1}^K x_i w_i \\ & \text{Subject to} && \sum_{i: l \in C_i} x_i = 1, && \forall l \in L \\ & && x_i \in \{0, 1\}, && 1 \leq i \leq K \end{aligned} \quad (3.5)$$

The algorithm to carry this out is given in algorithm 3.2. For our particular implementation, the igraph library [Csá+26] is used to generate the set of all cliques of a graph on line 4 and the linear program on line 9 is solved using lpSolve [CB24].

Lemma 3.11. *Algorithm 3.2 has $O(2^{2^m})$ time complexity.*

Proof. The graph with the most cliques for a given number of nodes m is the complete graph, having one clique for each subset of L , so $|\mathcal{B}| = O(2^m)$. Since $\mathcal{C} \subseteq \mathcal{B}$, then also $|\mathcal{C}| = O(2^m)$. Binary linear programming is exponential in the number of variables [Kar72, Main Theorem, Prob. 2] and we have a variable x_i for each element of $\mathcal{C}$ so the total runtime complexity is $O(2^{2^m})$. $\square$

Algorithm 3.2 Unsupervised Nominal Lumping

```

1: procedure LUMP_NOMINAL( $\hat{\mathbf{p}}, n, q, G_L$ )
2:    $\mathcal{C} \leftarrow []$  ▷ Find valid lumps
3:    $w \leftarrow []$ 
4:   for  $B_i \leftarrow \text{GET\_ALL\_CLIQUES}(G_L)$  do
5:     if  $N(B_i) \geq q$  then
6:       Append  $B_i$  to  $\mathcal{C}$ 
7:     end if
8:   end for
9:    $x \leftarrow \text{SOLVE\_BINARY\_LP} \left( \begin{array}{l} \text{Maximise } \sum_{i=1}^K x_i w_i \\ \text{Subject to } \sum_{i: l \in C_i} x_i = 1, \quad \forall l \in L \end{array} \right)$ 
10:    $K \leftarrow \text{length of } \mathcal{C}$  ▷ Reconstruct lumping from LP solution
11:    $L' \leftarrow []$ 
12:   for  $i \leftarrow 1$  to  $K$  do
13:     if  $x[i] = 1$  then
14:       Append  $\mathcal{C}[i]$  to  $L'$ 
15:     end if
16:   end for
17:   return  $L'$ 
18: end procedure

```

Algorithm 3.3 Unsupervised Hierarchical Lumping

```

1: procedure LUMP_HIERARCHICAL( $\hat{\mathbf{p}}, n, q, G_L$ )
2:    $L' \leftarrow []$ 
3:   for  $S \leftarrow \text{GET\_CONNECTED\_COMPONENTS}(G_L)$  do
4:      $n_S \leftarrow \sum_{i \in \text{NODES}(S)} D_i(L)$  ▷ Number of subjects in component  $S$ 
5:     Concatenate LUMP_NOMINAL( $\hat{\mathbf{p}}, n_S, q, S$ ) to  $L'$ 
6:   end for
7:   return  $L'$ 
8: end procedure

```

Hierarchical Variables One common case of nominal variables is those for which the preference sets are inherited from a hierarchy (e.g. provinces which can only be lumped with other provinces of the same country). In such a case, the preference graph is a disjoint union of the preference graphs within each ‘cluster’. A speedup over algorithm 3.2 can then be achieved by invoking it for each of these graph components separately, since the lumping of one cluster does not affect those of the others. This algorithm is given in algorithm 3.3.

Lemma 3.12. *Algorithm 3.3 has time complexity $O\left(\frac{m}{c}2^{2^c}\right)$, where c is the number of nodes in the largest component of the preference graph.*

Proof. The preference graph consists of at most m/c components of size c , which are clearly the slowest. Algorithm 3.2 is invoked for each of these components, of which the worst-case runtime is $O(2^{2^c})$, meaning the total time complexity is $O\left(\frac{m}{c}\right) \cdot O(2^{2^c}) = O\left(\frac{m}{c}2^{2^c}\right)$. $\square$

Algorithm 3.4 *Other Heuristic for Unsupervised Nominal Lumping*

```
1: procedure HEURISTICOTHER( $\hat{\mathbf{p}}, n, q, G_L$ )
2:    $L' \leftarrow L$ 
3:   while  $\exists l \in L' : N(l) < q$  do
4:      $curr \leftarrow \arg \min_{l \in L'} N(l)$ 
5:      $smallest\_neighbour \leftarrow \arg \min_{i \in \text{NEIGHBOURS}(l, G_L)} N(i)$ 
6:     Merge  $curr$  and  $smallest\_neighbour$  in  $L'$ 
7:     Merge  $curr$  and  $smallest\_neighbour$  in  $G_L$ 
8:   end while
9:   return  $L'$ 
10: end procedure
```

Notice that this is in general a significant speedup compared to $O(2^{2^m})$, since $c \leq m$ in general and $c \ll m$ whenever there are 2 or more clusters. The gap between c and m gets larger the more clusters there are.

Heuristics Given lemma 3.8 there exists a need for heuristics that are able to approximate the solution. Before we can do so, we must state lemma 3.13, which strongly caveats the usefulness of any heuristic.

Lemma 3.13. *Determining if there exists a lumping satisfying the constraints of problem 3.2 is NP-hard.*

Proof. Identical to lemma 3.8. □

Corollary 3.14. *Any polynomial-time approximation for problem 3.2 is not guaranteed to return a valid lumping.*

Corollary 3.15. *Problem 3.2 is inapproximable.*

Lump Small Levels Together Perhaps the simplest heuristic imaginable is to lump all levels that do not meet the threshold constraint together into one new level. This new level can then be understood as ‘other’, as is already practised [Wic25]. The added value of this heuristic, which is given in algorithm 3.4, is that it is able to take into account the preference graph, increasing flexibility. Of course, when a more complex preference graph is used, there is no guarantee that there will be one new level generated—which could easily be identified with ‘other’—so there will not be one unambiguous ‘other’ level in this instance.

Lump Small into Largest The obvious dual to algorithm 3.4 is to, instead of lumping levels together with their smallest neighbour, lump them with their largest neighbour. This is illustrated in algorithm 3.5. The intuitive idea behind this algorithm is that the ability of small levels to ‘pollute’ the observed effects of significantly larger levels is limited, so relatively little information is lost this way. The disadvantage is that when the preference graph is complete—as it often is—this algorithm ends up combining all levels not meeting the threshold into the largest, because the largest level before one lumping iteration is clearly still the largest level after this iteration (since it only grew), making for one large, very mixed and hard to interpret level.

Algorithm 3.5 *Largest Heuristic for Unsupervised Nominal Lumping*

```

1: procedure HEURISTICLARGEST( $\hat{\mathbf{p}}, n, q, G_L$ )
2:    $L' \leftarrow L$ 
3:   while  $\exists l \in L' : N(l) < q$  do
4:      $curr \leftarrow \arg \min_{l \in L'} N(l)$ 
5:      $largest\_neighbour \leftarrow \arg \max_{i \in \text{NEIGHBOURS}(l, G_L)} N(i)$ 
6:     Merge  $curr$  and  $largest\_neighbour$  in  $L'$ 
7:     Merge  $curr$  and  $largest\_neighbour$  in  $G_L$ 
8:   end while
9:   return  $L'$ 
10: end procedure

```

Smart Beyond these two simple heuristics, there also exists a smarter heuristic. The first component of this a greedy strategy, which iteratively takes the lumping step that has the lowest entropy cost immediately. An identical derivation to equation (3.2) reveals that this loss from lumping levels $a, b \in L$ together is

$$\Delta H(a, b) = (\hat{p}_a + \hat{p}_b) \log(\hat{p}_a + \hat{p}_b) - \hat{p}_a \log \hat{p}_a - \hat{p}_b \log \hat{p}_b. \quad (3.6)$$

The problem with a purely greedy approach is that it can lead to severely inadequate decisions, not just leading to suboptimal lumpings being found, but no lumping being found at all, when one does exist. Consider for instance the preference graph and samples given in figure 3.1a and with threshold $q = 3$. Using equation (3.6), lumping the two levels with 1 subject each together has entropy loss

$$\left(\frac{1}{6} + \frac{1}{6}\right) \log \left(\frac{1}{6} + \frac{1}{6}\right) - \frac{1}{6} \log \frac{1}{6} - \frac{1}{6} \log \frac{1}{6} \approx 0.231$$

and lumping together two levels, one with 1 subject and one with 2, has entropy loss

$$\left(\frac{1}{3} + \frac{1}{6}\right) \log \left(\frac{1}{3} + \frac{1}{6}\right) - \frac{1}{3} \log \frac{1}{3} - \frac{1}{6} \log \frac{1}{6} \approx 0.318.$$

A greedy algorithm will thus always choose to lump the two levels with 1 sample each together, yielding figure 3.1b. This is a problem, because it leaves every level having $2 < q$ samples and without any ability to continue lumping without violating the preference constraints, preventing it from finding the lumping in figure 3.1c.

This limitation is overcome by allowing the algorithm to ‘look ahead’. If combining the two lowest-cost levels together results in another level being unable to be lumped any more (due to the mutation of the preference graph following the lumping) and that level does not satisfy the threshold constraint yet, carrying out this lumping will lead the algorithm to get stuck in the manner of figure 3.1. As is shown in algorithm 3.6, the smart algorithm is able to detect these situations and avoid lumping as such. Of course, in light of corollary 3.14, this improvement can not eliminate the possibility of missing a possible lumping when it does exist whilst remaining computable in polynomial time, which was the motivation for developing a heuristic in the first place.

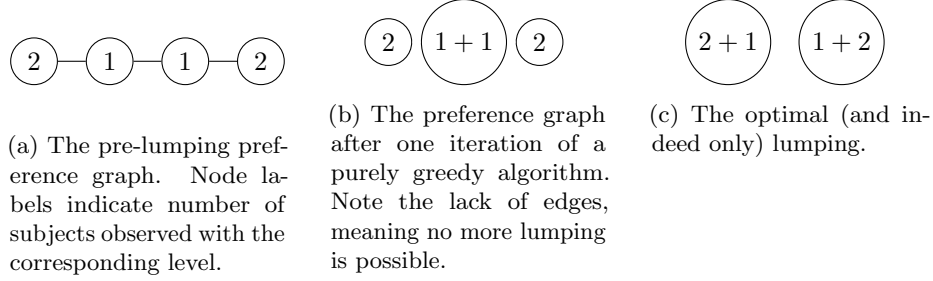


Figure 3.1: Illustration of the failure of a greedy approach.

Algorithm 3.6 *Smart* Heuristic for Unsupervised Nominal Lumping

```

1: procedure HEURISTICSMART( $\hat{\mathbf{p}}, n, q, G_L$ )
2:    $L' \leftarrow L$ 
3:   while  $\exists l \in L' : N(l) < q$  do
4:      $A \leftarrow \{(a, b) \in \text{EDGES}(G_L) \mid N(a) < q \vee N(b) < q\}$ 
5:     for  $(a, b) \in A$  do
6:        $\Delta H(a, b) \leftarrow (\hat{\mathbf{p}}[a] + \hat{\mathbf{p}}[b]) \log(\hat{\mathbf{p}}[a] + \hat{\mathbf{p}}[b]) - \hat{\mathbf{p}}[a] \log \hat{\mathbf{p}}[a] - \hat{\mathbf{p}}[b] \log \hat{\mathbf{p}}[b]$ 
7:        $B \leftarrow \{c \in (\text{NEIGHBOURS}(a, G_L) \cup \text{NEIGHBOURS}(b, G_L)) \setminus \{a, b\} : N(c) < q\}$ 
8:       for  $c \in B$  do
9:         if  $\text{NEIGHBOURS}(c, G_L) \subseteq \{a, b\}$  then
10:            $\Delta H(a, b) \leftarrow \infty$   $\triangleright$  Merging  $a, b$  would isolate  $c$ 
11:           break
12:         end if
13:       end for
14:     end for
15:      $(i, j) \leftarrow \arg \min_{(a, b) \in A} \Delta H(a, b)$ 
16:     if  $\Delta H(i, j) = \infty$  then
17:       fail
18:     end if
19:     Merge  $i$  and  $j$  in  $L'$ 
20:     Merge  $i$  and  $j$  in  $G_L$ 
21:   end while
22:   return  $L'$ 
23: end procedure

```

4 The Supervised Problem

In the previous section, not all information has been used. We have only looked at the variable being lumped itself, neglecting both the outcome of the regression model and the other variables. Including these should improve the ability of the post-lumping variable to explain the outcome, as more information has been taken into account. We will begin by exclusively looking at the outcome variable. That is, we will consider a variable Y being modelled as a function of our variable of interest X . When, for instance, two levels of X both have identical responses, both having exclusively $Y = 1$. Lumping these two levels together does not reduce the quality of the model, but the unsupervised algorithms might still avoid lumping them together. Considering the mutual information between the lumped variable and Y avoids this problem, by properly reflecting the fact that such a lumping preserves all the information in the data. After having treated outcomes, we will expand the outlook to also take into account the information presented by the other covariates of the model.

An important caveat to this section is that taking into account information about the outcome induces a substantial risk of overfitting. Any patterns observed in the sample can be idiosyncratic and not representative of the underlying data generation process. In practice, a number of other techniques, particularly in machine learning, suffer from a similar problem so a number of techniques which lay beyond the scope of this thesis have been developed. In particular k -fold cross-validation is designed to solve for this type of overfitting [Fus11].

4.1 Outcome-based Lumping

4.1.1 Discrete Outcomes

When the outcome is discrete, as it is in e.g. logistic regression, the problem is not meaningfully harder. We will see that the additive structure of the empirical mutual information lends itself very naturally to optimisation, using only slightly modified techniques from those in section 3.

Before it can be solved, the problem first needs to be stated precisely. As discussed before, the relevant metric is the mutual information between the lumped variable and the outcome, so it is sufficient to substitute the mutual information into problem 3.2, *mutatis mutandis*.

Problem 4.1. Given a categorical variable with levels L , a $q \in \mathbb{N}$, an outcome taking values in a finite $\mathcal{Y}$, and $n \geq m$ samples $(X_1, Y_1), \dots, (X_n, Y_n) \in L \times \mathcal{Y}$, find a lumping $L' = \{L'_1, \dots, L'_{m'}\}$ with maximal estimated information $\hat{J}_{\text{emp}}^{L'}(X; Y)$ such that:

1. For each $1 \leq i \leq m'$, $D_i(L') \geq q$, where $D_i(L') := \sum_{j=1}^n \mathbb{1}(X_j \in l'_i)$ is the number of samples having level l'_i under lumping L' .
2. If L is ordinal, each element of L' is contiguous.
3. If L is nominal, for each lumped level $A \in L'$, it holds that if $a, b \in A$, then $a \in P_b$.

Ordinal Case We proceed by analogy to section 3.2, beginning by deriving the information loss from lumping levels l_i through l_j together. For ease of

reading, we evaluate this loss for each of the terms in lemma and definition 2.20 individually.

- The loss in the entropy of X is already known. This is $J(i, j)$ from equation (3.2).
- The outcome does not experience a loss in entropy because the corresponding term, $\hat{H}_{\text{emp}}^{\mathcal{Y}}(Y)$, does not depend on $\mathcal{X}$ or X .
- The information loss of the joint distribution can be determined in the same way $J(i, j)$ was. Begin by expanding the pre-lumping entropy as

$$\hat{H}_{\text{emp}}^{L, \mathcal{Y}} = - \sum_{y \in \mathcal{Y}} \left(\sum_{k=i}^j \hat{p}_{l_k, y} \log \hat{p}_{l_k, y} + \sum_{\substack{1 \leq k < i \\ j < k \leq m}} \hat{p}_{l_k, y} \log \hat{p}_{l_k, y} \right).$$

The post-lumping entropy is

$$\hat{H}_{\text{emp}}^{L', \mathcal{Y}} = - \sum_{y \in \mathcal{Y}} \left(\left(\sum_{k=i}^j \hat{p}_{l_k, y} \right) \log \left(\sum_{k=i}^j \hat{p}_{l_k, y} \right) + \sum_{\substack{1 \leq k < i \\ j < k \leq m}} \hat{p}_{l_k, y} \log \hat{p}_{l_k, y} \right).$$

Taking the difference to obtain the loss, whilst noting the negative sign of the joint entropy term in lemma and definition 2.20, we find

$$- \left(\hat{H}_{\text{emp}}^{L, \mathcal{Y}} - \hat{H}_{\text{emp}}^{L', \mathcal{Y}} \right) = - \sum_{y \in \mathcal{Y}} \left(\left(\sum_{k=i}^j \hat{p}_{l_k, y} \right) \log \left(\sum_{k=i}^j \hat{p}_{l_k, y} \right) - \sum_{k=i}^j \hat{p}_{l_k, y} \log \hat{p}_{l_k, y} \right).$$

The total information loss from lumping levels l_i through l_j is then the sum of these three terms, or

$$K(i, j) := J(i, j) - \sum_{y \in \mathcal{Y}} \left(\left(\sum_{k=i}^j \hat{p}_{l_k, y} \right) \log \left(\sum_{k=i}^j \hat{p}_{l_k, y} \right) - \sum_{k=i}^j \hat{p}_{l_k, y} \log \hat{p}_{l_k, y} \right).$$

Just like in remark 3.4, $K(i, j)$ does not depend on levels not between l_i and l_j , meaning lemma 3.5 still holds. This means we can use dynamic program 3.3 and algorithm 3.1 to solve the supervised problem too, merely by replacing $J(i, j)$ with $K(i, j)$ on line 9.

Lemma 4.2. *The ordinal case of problem 4.1 can be solved in $O(m^3)$ time.*

Proof. Identical to lemma 3.6. $\square$

Nominal Case Just like for the ordinal case, the nominal case does not differ too much from the unsupervised problem, so we employ similar techniques. The supervised weights are, analogously to the unsupervised weights (3.4), assigned to each of the K valid cliques C_i of the preference graph:

$$v_i := w_i + \sum_{y \in \mathcal{Y}} \frac{N_y(C_i)}{n} \log \frac{N_y(C_i)}{n},$$

where $N_y(C_i)$ is the number of samples having $Y_i = y$ and $X_i \in C_i$, along with the same binary variables x_i indicating if the clique C_i is included in the lumping. It then holds that

$$\sum_{i=1}^K x_i v_i = \sum_{i=1}^K x_i w_i + \sum_{i=1}^K x_i \sum_{y \in \mathcal{Y}} \frac{N_y(C_i)}{n} \log \frac{N_y(C_i)}{n} = \hat{H}_{\text{emp}}^{L'} - \hat{H}_{\text{emp}}^{L', \mathcal{Y}}.$$

So that

$$\hat{I}_{\text{emp}}^{L', \mathcal{Y}} = \hat{H}_{\text{emp}}^{\mathcal{Y}} + \sum_{i=1}^K x_i v_i.$$

Just like for the ordinal case, the entropy of Y is not relevant since it is constant, so we obtain an integer linear program that is identical to equation (3.5), but with v_i instead of w_i :

$$\begin{aligned} & \text{Maximise} && \sum_{i=1}^K x_i v_i \\ & \text{Subject to} && \sum_{i: l \in C_i} x_i = 1, && \forall l \in L \\ & && x_i \in \{0, 1\}, && 1 \leq i \leq K \end{aligned}$$

For this reason, algorithm 3.2 can also be used in this instance by replacing the weights on line 9, meaning for the analysis of the runtime complexity, lemma 3.11 carries over. A further benefit of the reuse of the same algorithm is that the speedup for hierarchical variables (3.3) also carries over, improving performance for this problem too.

Lemma 4.3. *The nominal case of problem 4.1 can be solved in $O(2^{2^m})$ time.*

Proof. Identical to lemma 3.11. $\square$

4.1.2 Continuous Outcomes

Using the Ross estimator (2.23), we can reformulate problem 4.1 to be applicable to continuous outcome variables.

Problem 4.4. Given a categorical variable with levels L , a $q \in \mathbb{N}$, an outcome taking values in a $\mathcal{Y} \subseteq \mathbb{R}$, and $n \geq m$ samples $(X_1, Y_1), \dots, (X_n, Y_n) \in L \times \mathcal{Y}$, find a lumping $L' = \{L'_1, \dots, L'_{m'}\}$ with maximal estimated information $\hat{I}_{\text{Ross}}^{L'}(X; Y)$ such that:

1. For each $1 \leq i \leq m'$, $D_i(L') \geq q$, where $D_i(L') := \sum_{j=1}^n \mathbb{1}(X_j \in l'_i)$ is the number of samples having level l'_i under lumping L' .
2. If L is ordinal, each element of L' is contiguous.
3. If L is nominal, for each lumped level $A \in L'$, it holds that if $a, b \in A$, then $a \in P_b$.

What is immediately apparent with regards to solving problem 4.4 is that the term $\psi(n) + \psi(k)$ is a constant with respect to the lumping process, so it can safely be ignored during optimisation. This leaves the reward function as

$$-\frac{1}{n} \sum_{i=1}^n (\psi(n_i^{L'}) + \psi(c_i^{L'})) = -\frac{1}{n} \left(\sum_{i=1}^n \psi(n_i^{L'}) + \sum_{i=1}^n \psi(c_i^{L'}) \right). \quad (4.1)$$

Ordinal Case We once again proceed to identify the effect of lumping levels l_i through l_j . Call the lumping obtained from this L' . Notice that if $X_s \notin \{l_i, \dots, l_j\}$ then both $n_s^L = n_s^{L'}$ and $c_s^L = c_s^{L'}$ so we need only concern ourselves with those terms of equation (4.1) corresponding to data points taking values in $\{l_i, \dots, l_j\}$. Call this set $S = \{s : X_s \in \{l_i, \dots, l_j\}\}$. For each $s \in S$ we can now look at the loss in both terms:

- The value of $n_s^{L'}$ is just the total number of samples having a value in $\{l_i, \dots, l_j\}$, which is precisely the $C(i, j)$ from equation (3.3). This implies the contribution from the first term in equation (4.1) is

$$-\frac{1}{n} \sum_{s \in S} \psi(C(i, j)) = -\frac{1}{n} |S| \psi(C(i, j)) = -\frac{1}{n} C(i, j) \psi(C(i, j)).$$

Since the term before lumping was

$$\begin{aligned} -\frac{1}{n} \sum_{s \in S} \psi(n_s^L) &= -\frac{1}{n} \sum_{t=i}^j \sum_{X_s=l_t} \psi(n_s^L) \\ &= -\frac{1}{n} \sum_{t=i}^j \sum_{X_s=l_t} \psi(D_t(L)) \\ &= -\frac{1}{n} \sum_{t=i}^j D_t(L) \psi(D_t(L)), \end{aligned}$$

the loss is

$$\frac{1}{n} C(i, j) \psi(C(i, j)) - \frac{1}{n} \sum_{t=i}^j D_t(L) \psi(D_t(L)) \quad (4.2)$$

- For the second term, the value of $c_s^{L'}$ cannot be ascertained as easily, since it relies on the k^{th} closest element to Y_s in the set $\{Y_t : X_t \in \{l_i, \dots, l_j\}\}$, the behaviour of which is hard to control or predict as i and j change. For this reason, it is simply recomputed at each iteration and the associated loss is

$$\frac{1}{n} \sum_{s \in S} \psi(c_s^{L'}) - \frac{1}{n} \sum_{s \in S} \psi(c_s^L). \quad (4.3)$$

Summing equations (4.2) and (4.3) results in the total information loss from lumping levels l_i through l_j ,

$$\begin{aligned} R(i, j) &:= \frac{1}{n} C(i, j) \psi(C(i, j)) - \frac{1}{n} \sum_{t=i}^j D_t(L) \psi(D_t(L)) + \frac{1}{n} \sum_{s \in S} \psi(c_s^{L'}) - \frac{1}{n} \sum_{s \in S} \psi(c_s^L) \\ &= \frac{1}{n} \left(C(i, j) \psi(C(i, j)) - \sum_{t=i}^j D_t(L) \psi(D_t(L)) + \sum_{s \in S} \left(\psi(c_s^{L'}) - \psi(c_s^L) \right) \right). \end{aligned}$$

We once again note that, since $R(i, j)$ does not depend on any levels not between l_i and l_j , we can cross-apply lemma 3.5 and use dynamic program 3.3. Consequently, algorithm 3.1 can also be used, substituting for $J(i, j)$ the new loss function $R(i, j)$, caveating that algorithm 4.1 is used to calculate the loss.

Algorithm 4.1 Calculation of Cost Function for Continuous Supervised Ordinal Lumping

```

1: procedure CONTINUOUSCOST( $i, j, \mathbf{X}, \mathbf{Y}$ )
2:    $n \leftarrow \text{length of } \mathbf{X}$ 
3:    $res \leftarrow C(i, j)\psi(C(i, j)) - \sum_{t=i}^j D_t(L)\psi(D_t(L))$ 
4:   for  $s \in S$  do
5:      $r \leftarrow |\mathbf{Y} - \mathbf{Y}[s]|$  ▷ Absolute value is elementwise
6:      $\mathbf{X}', r' \leftarrow \text{sort } \mathbf{X}, r \text{ ascendingly on } r$ 
7:      $d_s^L \leftarrow r'[\mathbf{X}' = \mathbf{X}[s]][k+1]$  ▷  $k+1$  to account for the fact  $s$  itself
       should not be counted
8:      $c_s^L \leftarrow \max \{l : r'[l] \leq d_s^L\}$ 
9:      $d_s^{L'} \leftarrow r'[\mathbf{X}' \in \{l_i, \dots, l_j\}][k+1]$ 
10:     $c_s^{L'} \leftarrow \max \{l : r'[l] \leq d_s^{L'}\}$ 
11:     $res \leftarrow res + \psi(c_s^{L'}) - \psi(c_s^L)$ 
12:  end for
13:  return  $\frac{1}{n}res$ 
14: end procedure

```

Lemma 4.5. *The ordinal case of problem 4.4 can be solved in $O(m^2 n^2 \log n)$ time.*

Proof. We know from the proof of lemma 3.6 that algorithm 4.1 will be invoked $O(m^2)$ times. The sorting on line 6 can be done in $O(n \log n)$ time [Ala+24] and is reached $|S| = O(n)$ times per call of algorithm 4.1. In total this results in a complexity of $O(m^2) \cdot O(n) \cdot O(n \log n) = O(m^2 n^2 \log n)$. $\square$

Nominal Case Just like in section 4.1.1, we proceed by assigning to each of the K valid cliques C_i of the preference graph a weight

$$u_i := -\frac{N(C_i)}{n}\psi(N(C_i)) - \frac{1}{n} \sum_{X_j \in C_i} \psi(c_j^{L'}),$$

where L' denotes the lumping obtained by combining all levels in C_i . Introducing indicator variables x_i to denote if C_i is included in the optimal lumping, we see the same result as before.

Lemma 4.6.

$$\sum_{i=1}^K x_i u_i = -\frac{1}{n} \left(\sum_{i=1}^n \psi(n_i^{L'}) + \sum_{i=1}^n \psi(c_i^{L'}) \right).$$

Proof. We begin by looking at the first term, corresponding to $n_i^{L'}$. Using a

similar derivation to equation (4.2), we see

$$\begin{aligned}
-\frac{1}{n} \sum_{i=1}^n \psi(n_i^{L'}) &= -\frac{1}{n} \sum_{C_i \in L'} \sum_{X_j \in C_i} \psi(n_i^{L'}) \\
&= -\frac{1}{n} \sum_{C_i \in L'} \sum_{X_j \in C_i} \psi(N(C_i)) \\
&= -\frac{1}{n} \sum_{C_i \in L'} N(C_i) \psi(N(C_i)) \\
&= \sum_{C_i \in L'} -\frac{1}{n} N(C_i) \psi(N(C_i)) \\
&= \sum_{C_i \in L'} -x_i \frac{1}{n} N(C_i) \psi(N(C_i)),
\end{aligned}$$

where the last equality follows from the fact that, by construction, $x_i = 1$ whenever $C_i \in L'$. For the same reason, the $c_j^{L'}$ terms have

$$-\frac{1}{n} \sum_{i=1}^n \psi(c_i^{L'}) = -\frac{1}{n} \sum_{C_i \in L'} \sum_{X_j \in C_i} \psi(c_i^{L'}) = \sum_{C_i \in L'} -x_i \frac{1}{n} \sum_{X_j \in C_i} \psi(c_j^{L'}).$$

Summing these two equalities shows that

$$\sum_{C_i \in L'} x_i u_i = -\frac{1}{n} \left(\sum_{i=1}^n \psi(n_i^{L'}) + \sum_{i=1}^n \psi(c_i^{L'}) \right).$$

The lemma then follows from the fact that all other terms in $\sum_{i=1}^K x_i u_i$ have $x_i = 0$ and thus must be zero. $\square$

This implies the following integer linear program:

$$\begin{aligned}
&\text{Maximise} && \sum_{i=1}^K x_i u_i \\
&\text{Subject to} && \sum_{i: l \in C_i} x_i = 1, && \forall l \in L \\
&&& x_i \in \{0, 1\}, && 1 \leq i \leq K
\end{aligned}$$

Which can once again be solved using algorithm 3.2. Note that unlike in previous cases, the benefit of the speedup for hierarchical variables in lemma 3.12 does not cross-apply, because the value of $c_i^{L'}$ depends, via $d_i^{L'}$, on the values in different components as well.

Lemma 4.7. *The nominal case of problem 4.4 can be solved in $O(2^{2^m})$ time.*

Proof. Identical to lemma 3.11. $\square$

4.2 Context

Now, we consider the full context provided by the other covariates of our regression model $Z = f(X, Y) + \varepsilon$, with Z the outcome, X the variable being

lumped, and Y any other covariates being taken into account. Notice that we then have 2 natural choices for the quantity of interest,

$$I(X', Y; Z),$$

which tries to optimise the amount of information X' and Y contain about Z together and

$$I(X'; Z|Y),$$

which optimises the information X' adds to the problem, given the other covariates. Whilst the interpretation is subtly different, lemma 4.8 tells us we do not need to concern ourselves with the distinction.

Lemma 4.8.

$$I(X', Y; Z) = H(Z) - H(Z|Y) + I(X'; Z|Y).$$

Proof. Applying lemma 2.9 twice yields

$$\begin{aligned} I(X', Y; Z) &= H(Z) - H(Z|X', Y) \\ &= H(Z) - H(Z|Y) + H(Z|Y) - H(Z|X', Y) \\ &= H(Z) - H(Z|Y) + I(X'; Z|Y) \end{aligned} \quad \square$$

Notice that $H(Z)$ and $H(Z|Y)$ are both independent of X' so, for the lumping problem, optimising $I(X'; Z|Y)$ and $I(X', Y; Z)$ are equivalent. When all these quantities are discrete, this problem is already solved by section 4.1.1, by simply imposing a preference graph on (X', Y) which disallows the combining of any of the levels of Y . When one of Y or Z is continuous (including when Y consists of multiple covariates, some of which continuous and some of which discrete) the problem of numeric integration outlined in section 2.4 becomes even stronger, since the data has the ability to be even more sparse the more continuous dimensions there are, meaning no general solution for these cases has been found, as the trick used for the context-free supervised case, in which the estimator splits additively can no longer be applied.

5 Numerical Evaluation

Having established the algorithms in sections 3 and 4, we now turn to evaluate the performance of lumping. We will begin by doing a basic feasibility test, to see if the program runs quickly enough to be used in practice. Then, we will check how stable the algorithms are by checking how sensitive they are to small modifications in the input data, as well as verifying how well the heuristics approximate the optimal value. Finally, having established that the algorithms are usable using the previous two experiments, we will examine the effect lumping has on the quality of models.

5.1 Benchmark

In light of all the theoretical runtime complexities derived in sections 3 and 4, it is worthwhile to collect actual data on the runtime of the algorithms. These execution times are given in figure 5.1. Note that out of the supervised lumping functions, only the continuous ordinal case has been included as a separate benchmark. This is because all of the other supervised lumping functions are merely reimplementations of their unsupervised analogues, with a tweaked reward function, so their performance is largely identical to their unsupervised versions. The continuous ordinal function is worth benchmarking separately since the introduction of algorithm 4.1 changes the runtime complexity, as per lemma 4.5.

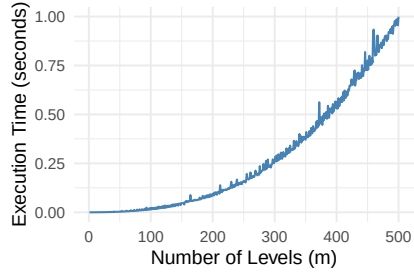
We observe that indeed, as predicted by lemmas 3.6 and 3.11, ordinal variables are computationally a lot easier than nominal variables. Indeed, ordinal variables can be lumped in under a second up to about $m = 500$, which is more than enough levels for most applications, but nominal variables take over a minute with only 13 levels. For this reason, it is good that the heuristic is significantly more performant, as indicated by figure 5.1c.

Figure 5.2 shows the performance of the hierarchical optimisation in algorithm 3.3. The first thing of note is that this confirms lemma 3.12, as the runtime grows rapidly when growing the maximum component size, but only moderately for larger numbers of levels. Beyond these asymptotic results, we also see that the overall execution time is consistently lower than the full nominal solver (5.1b), being able to easily handle up to 200 levels, depending on the size of the components.

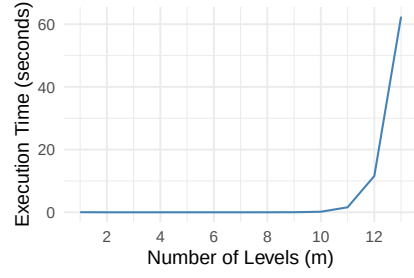
5.2 Stability

It is important to ascertain how sensitive the lumping process is to random fluctuations in the data. To find this, we use confidence intervals on the final information remaining after lumping is complete, obtained by bootstrapping 5000 samples from a synthetic dataset (A.6). The threshold parameter q is allowed to vary in order to show the effect of the threshold on the stability of the process.⁹ The nominal variables are given a complete graph as their preference graphs. The hierarchical lumping functions are excluded, since the nature of their preference graphs makes it so that they often do not admit a valid lumping, where a complete preference graph would. No knowledge is lost

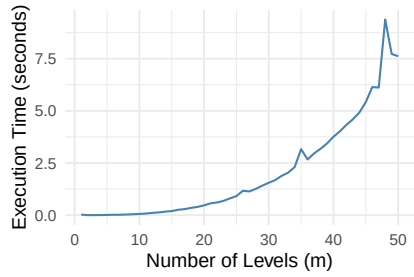
⁹Note that q/n is not allowed to run all the way from zero to one. This is because whenever $q > n/2$, the pigeonhole principle implies only one lumping exists: the one in which all levels are lumped into one level. This lumping clearly has zero informational content.



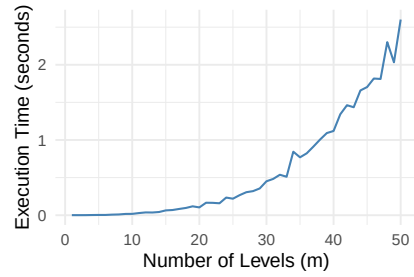
(a) Unsupervised lumping of an ordinal variable. Generated using listing A.1.



(b) Unsupervised lumping of a nominal variable. Generated using listing A.2.



(c) Unsupervised lumping of a nominal variable using the smart heuristic. Generated using listing A.3.



(d) Continuous supervised lumping of an ordinal variable. Generated using listing A.4.

Figure 5.1: Benchmarks of the lumping functions. Note the different axis scales.

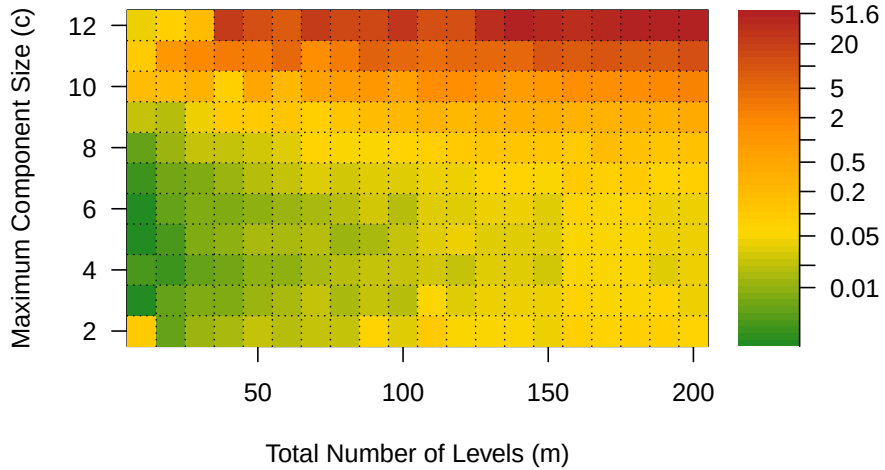


Figure 5.2: Benchmark results for hierarchical variables. Heatmap shows effect of both the number of levels m and the size of the largest component c on a log scale. Generated using listing A.5.

from this exclusion, as, because they are merely a more performant version of the general nominal functions, the distributions of their post-lumping informations are identical. These confidence intervals are given in figure 5.3.

Figures 5.3a to 5.3c show that the unsupervised methods have good, tight confidence intervals. By contrast, the supervised methods in figures 5.3d and 5.3e have substantially wider intervals, even including zero for every value of q . This is unsurprising, given the nominal functions are not only affected by the variance in the sampling of the covariate, but also by that of the outcome.

Another conclusion that can be drawn from figure 5.3 is that for moderate values¹⁰ of q , the decrease in information content has a very gentle slope. This is good, as it implies lumping can be used, without destroying a large amount of information. Furthermore, it suggests that one need not worry too much about the exact value of q as it is not too sensitive to small perturbations.

5.3 Quality of Heuristics

Corollary 3.14 puts a limit on the theoretical performance of any heuristic. It, however, only says that no algorithm can *in general* approximate the optimal solution well and does not bar heuristics from performing well in a large number of cases. We therefore also investigate the performance of the heuristics. This is particularly motivated by the strong similarity between the confidence intervals for the exact solution in figure 5.3b and that for the heuristic (5.3c).

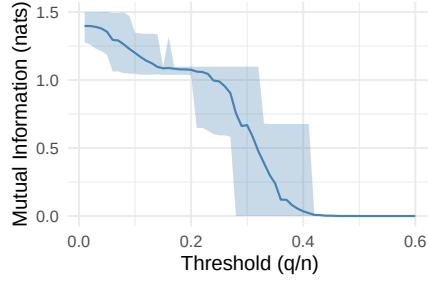
We check this by looking at the difference between the information preserved by the optimal solution and that preserved by the heuristics. Once again, we consider a complete preference graph and let q take on a range of values and bootstrap 5000 samples to obtain confidence intervals. The results of this experiment are given in figure 5.4.

The first thing worth noting is that the largest heuristic performs a lot worse than the other two, which perform nearly identically. The smart and other heuristics are in fact so good that their 95% confidence interval is often $[0, 0]$. The places where they do have bumps (around $q/n = 0.2$ and $q/n = 0.45$) are exactly the same thresholds where the information of the optimal solution goes through a rapid fall and an increase in variance (see figure 5.3b). These are the places where the balance is such that some of the bootstrap samples need one low-cost lumping and some others need a more costly lumping, meaning that the samples drawn are close to the tipping point between these two lumpings, making it more difficult for the heuristics to distinguish if the more informational lumping is still valid or not.

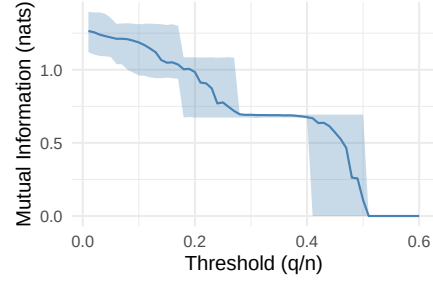
5.4 Model Improvement

The final check we conduct is on the quality of the post-lumping model compared to the pre-lumping model. To do this, we once again generate a synthetic dataset (A.13) and bootstrap 1000 samples from it. For each sample, a regression model is fit, then the lumping is applied and a new model is fit. The errors in the predictions both of these models make outside of their training data are then

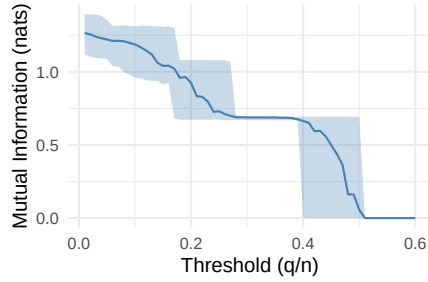
¹⁰In this example up to about 0.25 for ordinal variables, corresponding to when the lumps “Very Low+Low” and “High+Very High”, which both have probability 0.3, start having random variations cause them to sometimes not meet the threshold frequently. For nominal variables, this transition occurs around $q/n = 0.2$ for similar reasons.



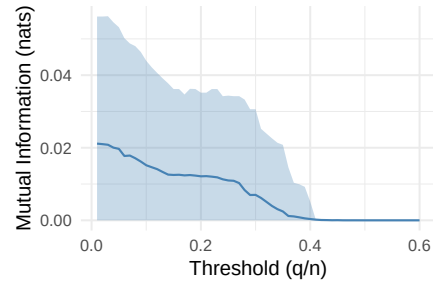
(a) Unsupervised lumping of an ordinal variable. Generated using listing A.7.



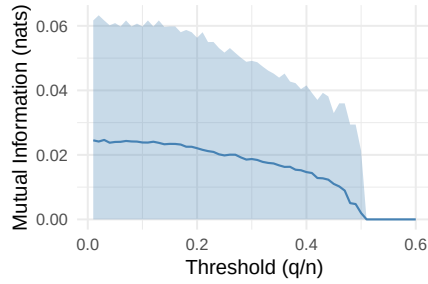
(b) Unsupervised lumping of a nominal variable. Generated using listing A.8.



(c) Unsupervised lumping of a nominal variable using the smart heuristic. Generated using listing A.9.



(d) Supervised lumping of an ordinal variable. Generated using listing A.10.



(e) Supervised lumping of a nominal variable. Generated using listing A.11.

Figure 5.3: 95% confidence intervals for mutual information preserved after lumping variables to various threshold values q . The dark blue line indicates the mean information and the light blue region is the confidence interval.

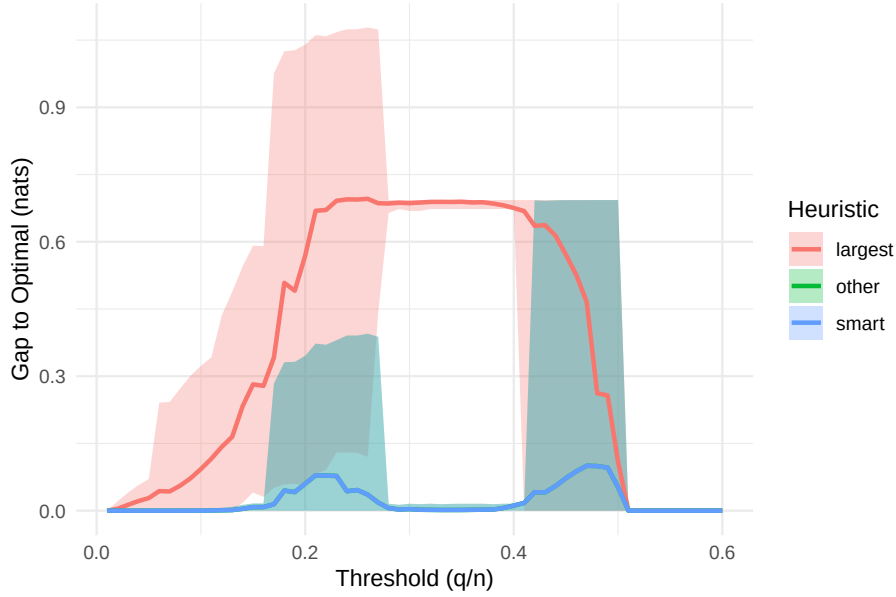
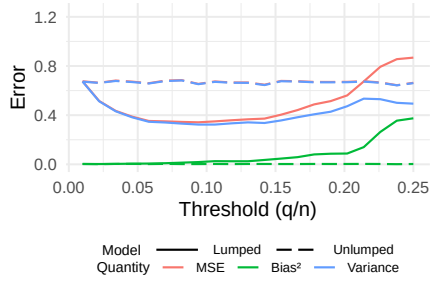


Figure 5.4: Mean differences between the optimal solution and the heuristics. 95% confidence intervals indicated by shading. Generated using listing A.12.

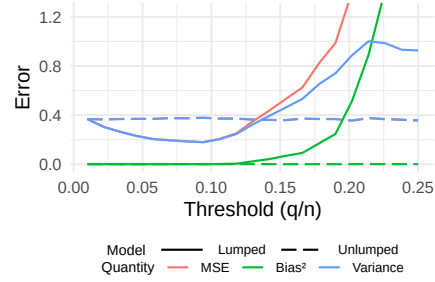
recorded. Just like before, the value of q is also allowed to vary. The quantity compared is the mean squared error (MSE). The results for each type of lumping method are shown in figure 5.5.

The main result of note here is that, given an appropriate value of q , the model using lumped data has a lower MSE than that using unlumped data, indicating an improvement from applying the technique. Notably, this reduction is the result of a large reduction in variance, at the expense of a small amount of bias—which is exactly what should be expected from a regularisation method.

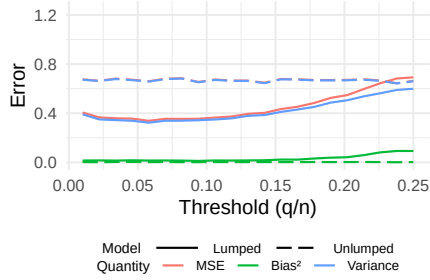
A further point of note is that the values of q yielding the best results are similar to those found to be relatively stable in figure 5.3, indicating the same range around $q/n \approx 0.05$. Around this value there is zero to no increase in bias, which only begins to pick up later.



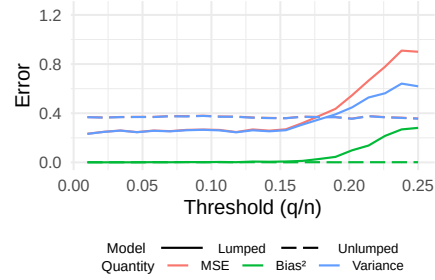
(a) Unsupervised lumping of an ordinal variable. Generated using listing A.14.



(b) Unsupervised lumping of a nominal variable. Generated using listing A.15.



(c) Supervised lumping of an ordinal variable. Generated using listing A.16.



(d) Supervised lumping of a nominal variable. Generated using listing A.17.

Figure 5.5: The performance of models before and after lumping. Colour indicates the quantity, being the bias squared, variance, and their sum, the MSE and line type indicates if it is the lumped model or the unlumped model. Note that the bias of the unlumped model is usually (near-)zero, meaning the lines for the unlumped variance and MSE largely coincide. Confidence intervals omitted to avoid visual clutter.

6 Conclusion

Lumping has been shown to be a method for the regularisation of various models. The use of a threshold constraint leads to the eradication of small categories. Further constraints are imposed on what lumpings are valid: contiguity in the case of ordinal variables, and respect for the user-specified rules as to what pairings are allowed in the case of nominal variables. We have seen that these constraints give rise to vastly different behaviour; nominal variables have superexponential runtime complexity, whereas ordinal variables, due to their requirement that lumps be contiguous, cut down the number of lumpings vastly, enabling polynomial-time algorithms. These bounds have also been experimentally verified in a benchmark. These benchmarks show that, perhaps through the use of the heuristics for the nominal case, it is computationally feasible to use lumping on real data.

Combining this computational feasibility with the fact that lumping is experimentally shown to be relatively resistant to small perturbations of the input data and the fact that it can induce a reduction in MSE means that lumping is a highly practical method for dimensionality reduction.

References

- [AK01] A. Antos and I. Kontoyiannis. “Convergence Properties of Functional Estimates for Discrete Distributions”. In: *Random Structures & Algorithms* 19 (Sept. 2001). DOI: 10.1002/rsa.10019.
- [Ala+24] M. A. Ala’Anzy et al. “Comparative Analysis of Sorting Algorithms: A Review”. In: *2024 11th International Conference on Soft Computing & Machine Intelligence (ISCMi)*. 2024, pp. 88–100. DOI: 10.1109/ISCMi63661.2024.10851593.
- [BCL92] J.-P. Barthélemy, G. Cohen, and A. Lobstein. *Complexité algorithmique et problèmes de communications*. Collection Technique et Scientifique des Télécommunications. Paris: Masson, 1992. ISBN: 2-225-82672-2.
- [BLC02] D. Barbará, Y. Li, and J. Couto. “COOLCAT: an entropy-based algorithm for categorical clustering”. In: *Proceedings of the Eleventh International Conference on Information and Knowledge Management*. Association for Computing Machinery, 2002, pp. 582–589. DOI: 10.1145/584792.584888.
- [CB24] G. Csárdi and M. Berkelaar. *lpSolve: Interface to 'Lp_solve' v. 5.5 to Solve Linear/Integer Programs*. R package version 5.6.23. 2024. DOI: 10.32614/CRAN.package.lpSolve. URL: <https://github.com/gaborcsardi/lpSolve>.
- [Csá+26] G. Csárdi et al. *igraph: Network Analysis and Visualization in R*. R package version 2.3.0.9001. 2026. DOI: 10.5281/zenodo.7682609. URL: <https://CRAN.R-project.org/package=igraph>.
- [CT91] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. 1st ed. Wiley Series in Telecommunications. New York: John Wiley & Sons, 1991. ISBN: 0-471-06259-6.
- [DKS95] J. Dougherty, R. Kohavi, and M. Sahami. “Supervised and unsupervised discretization of continuous features”. In: *Machine learning proceedings 1995*. Elsevier, 1995, pp. 194–202. DOI: 10.1016/B978-1-55860-377-6.50032-3.
- [Fus11] T. Fushiki. “Estimation of prediction error by using K-fold cross-validation”. In: *Statistics and Computing* 21.2 (2011), pp. 137–146. DOI: 10.1007/s11222-009-9153-8.
- [Gra11] R. M. Gray. *Entropy and Information Theory*. 2nd ed. New York: Springer, 2011. ISBN: 978-1-4419-7969-8.
- [GW14] G. Grimmett and D. Welsh. *Probability. An introduction*. 2nd ed. Oxford: Oxford University Press, 2014. ISBN: 978-0-19-870997-8.
- [Har01] F. Harrell. *Regression Modeling Strategies*. 1st ed. 2001. ISBN: 978-1-4419-2918-1.
- [HS09] J. Hausser and K. Strimmer. “Entropy Inference and the James-Stein Estimator, with Application to Nonlinear Gene Association Networks”. In: *J. Mach. Learn. Res.* 10 (Dec. 2009), pp. 1469–1484.

- [HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009. ISBN: 9780387848846.
- [IK77] S. Iyanaga and Y. Kawada. *Encyclopedic Dictionary of Mathematics*. 2nd ed. MIT Press, 1977. ISBN: 0-262-59010-7.
- [Jen06] J. L. W. V. Jensen. “Sur les fonctions convexes et les inégalités entre les valeurs moyennes”. In: *Acta Mathematica* 30 (1906), pp. 175–193. DOI: 10.1007/bf02418571.
- [Kar72] R. M. Karp. “Reducibility among Combinatorial Problems”. In: *Complexity of Computer Computations*. Ed. by R. E. Miller, J. W. Thatcher, and J. D. Bohlinger. Boston, MA: Springer US, 1972, pp. 85–103. ISBN: 978-1-4684-2001-2. DOI: 10.1007/978-1-4684-2001-2_9.
- [KH06] A. J. Knobbe and E. K. Y. Ho. “Maximally informative k-itemsets and their efficient discovery”. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2006, pp. 237–244. DOI: 10.1145/1150402.1150431.
- [KSG04] A. Kraskov, H. Stögbauer, and P. Grassberger. “Estimating mutual information”. In: *Phys. Rev. E* 69 (6 June 2004), p. 066138. DOI: 10.1103/PhysRevE.69.066138.
- [Pan03] L. Paninski. “Estimation of Entropy and Mutual Information”. In: *Neural Computation* 15.6 (June 2003). DOI: 10.1162/089976603321780272.
- [Ros14] B. Ross. “Mutual Information between Discrete and Continuous Data Sets”. In: *PloS one* 9 (Feb. 2014), e87357. DOI: 10.1371/journal.pone.0087357.
- [Wic25] H. Wickham. *forcats: Tools for Working with Categorical Variables (Factors)*. R package version 1.0.1. 2025. URL: <https://forcats.tidyverse.org/>.

A Code

Please note that this appendix only contains the code used to generate the figures and data in this thesis. The package itself can be obtained at <https://github.com/daankoning/optLump/releases/tag/v1.0.0>.

```
1 library(optLump)
2 library(ggplot2)
3
4 set.seed(0)
5
6 args <- commandArgs(trailingOnly = TRUE)
7
8 cache_path <- "figures/ordinal_benchmark_cache.csv"
9 output_path <- "figures/ordinal_benchmark.pdf"
10
11 if ("--rebuild" %in% args || !file.exists(cache_path)) {
12   m_value <- 1:500
13   execution_time <- numeric(length(m_value))
14   for (i in seq_along(m_value)) {
15     m <- m_value[i]
16     print(m)
17
18     counts <- sample(5:15, m, replace = TRUE)
19     names(counts) <- paste0("L", 1:m)
20
21     threshold <- sum(counts) / 3
22
23     start_time <- Sys.time()
24     maximum_mutual_information_ordinal(counts, threshold)
25     end_time <- Sys.time()
26
27     execution_time[i] <- as.numeric(difftime(end_time, start_time,
28       units = "secs"))
29   }
30
31   plot_data <- data.frame(m_value, execution_time)
32   write.csv(plot_data, cache_path, row.names = FALSE)
33 } else {
34   plot_data <- read.csv(cache_path)
35 }
36
37 plot_data |>
38   ggplot(aes(x = m_value)) +
39   geom_line(aes(y=execution_time), color = 'steelblue', linewidth =
40     1) +
41   theme_minimal(base_size = 20) +
42   labs(
43     x = "Number of Levels (m)",
44     y = "Execution Time (seconds)"
45   )
46
47 ggsave(output_path, width = 6, height = 4)
```

Listing A.1: ordinal`benchmark.R

```
1 library(optLump)
2 library(ggplot2)
3
4 set.seed(0)
5
6 args <- commandArgs(trailingOnly = TRUE)
```

```

7
8 cache_path <- "figures/nominal_benchmark_cache.csv"
9 output_path <- "figures/nominal_benchmark.pdf"
10
11 if ("--rebuild" %in% args || !file.exists(cache_path)) {
12   m_value <- 1:13
13   execution_time <- numeric(length(m_value))
14   for (i in seq_along(m_value)) {
15     m <- m_value[i]
16     print(m)
17
18     counts <- sample(5:15, m, replace = TRUE)
19     names(counts) <- paste0("L", 1:m)
20
21     threshold <- sum(counts) / 3
22
23     start_time <- Sys.time()
24     maximum_mutual_information_nominal(counts, threshold)
25     end_time <- Sys.time()
26
27     execution_time[i] <- as.numeric(difftime(end_time, start_time,
28       units = "secs"))
29   }
30
31   plot_data <- data.frame(m_value, execution_time)
32   write.csv(plot_data, cache_path, row.names = FALSE)
33 } else {
34   plot_data <- read.csv(cache_path)
35 }
36
37 plot_data |>
38   ggplot(aes(x = m_value)) +
39   geom_line(aes(y=execution_time), color = 'steelblue', linewidth =
40     1) +
41   scale_x_continuous(breaks = seq(0, 14, by = 2)) +
42   theme_minimal(base_size = 20) +
43   labs(
44     x      = "Number of Levels (m)",
45     y      = "Execution Time (seconds)"
46   )
47
48 ggsave(output_path, width = 6, height = 4)

```

Listing A.2: nominal benchmark.R

```

1 library(optLump)
2 library(ggplot2)
3
4 set.seed(0)
5
6 args <- commandArgs(trailingOnly = TRUE)
7
8 cache_path <- "figures/heuristic_benchmark_cache.csv"
9 output_path <- "figures/heuristic_benchmark.pdf"
10
11 if ("--rebuild" %in% args || !file.exists(cache_path)) {
12   m_value <- 1:50
13   execution_time <- numeric(length(m_value))
14   for (i in seq_along(m_value)) {
15     m <- m_value[i]
16     print(m)
17
18     counts <- sample(5:15, m, replace = TRUE)

```

```

19     names(counts) <- paste0("L", 1:m)
20
21     threshold <- sum(counts) / 3
22
23     start_time <- Sys.time()
24     maximum_mutual_information_nominal_heuristic(counts, threshold)
25     end_time <- Sys.time()
26
27     execution_time[i] <- as.numeric(difftime(end_time, start_time,
28       units = "secs"))
29   }
30   plot_data <- data.frame(m_value, execution_time)
31   write.csv(plot_data, cache_path, row.names = FALSE)
32 } else {
33   plot_data <- read.csv(cache_path)
34 }
35
36 plot_data |>
37   ggplot(aes(x = m_value)) +
38     geom_line(aes(y=execution_time), color = 'steelblue', linewidth =
39       1) +
40     theme_minimal(base_size = 20) +
41     labs(
42       x = "Number of Levels (m)",
43       y = "Execution Time (seconds)"
44     )
45 ggsave(output_path, width = 6, height = 4)

```

Listing A.3: heuristic benchmark.R

```

1 library(optLump)
2 library(ggplot2)
3
4 set.seed(0)
5
6 args <- commandArgs(trailingOnly = TRUE)
7
8 cache_path <- "figures/ordinal_cont_sup_benchmark_cache.csv"
9 output_path <- "figures/ordinal_cont_sup_benchmark.pdf"
10
11 if ("--rebuild" %in% args || !file.exists(cache_path)) {
12   m_value <- 1:50
13   execution_time <- numeric(length(m_value))
14   for (i in seq_along(m_value)) {
15     m <- m_value[i]
16     print(m)
17
18     counts <- sample(5:15, m, replace = TRUE)
19     n <- sum(counts)
20     x <- factor(rep(paste0("L", 1:m), counts), levels = paste0("L",
21       1:m), ordered = TRUE)
22     y <- rnorm(n)
23
24     threshold <- n / 3
25
26     start_time <- Sys.time()
27     maximum_mutual_information_ordinal_supervised_continuous(x, y,
28       threshold)
29     end_time <- Sys.time()
30   }
31 }

```

```

29     execution_time[i] <- as.numeric(difftime(end_time, start_time,
30       units = "secs"))
31   }
32   plot_data <- data.frame(m_value, execution_time)
33   write.csv(plot_data, cache_path, row.names = FALSE)
34 } else {
35   plot_data <- read.csv(cache_path)
36 }
37
38 plot_data |>
39   ggplot(aes(x = m_value)) +
40     geom_line(aes(y = execution_time), color = 'steelblue', linewidth
41       = 1) +
42     theme_minimal(base_size = 20) +
43     labs(
44       x = "Number of Levels (m)",
45       y = "Execution Time (seconds)"
46     )
47 ggsave(output_path, width = 6, height = 4)

```

Listing A.4: ordinal'cont'sup'benchmark.R

```

1 library(optLump)
2
3 set.seed(0)
4
5 args <- commandArgs(trailingOnly = TRUE)
6
7 cache_path <- "figures/hierarchical_benchmark_cache.csv"
8 output_path <- "figures/hierarchical_benchmark.pdf"
9
10 m_values <- seq(10, 200, by = 10)
11 cluster_sizes <- 2:12
12
13 if ("--rebuild" %in% args || !file.exists(cache_path)) {
14   data <- matrix(NA, nrow = length(m_values), ncol = length(cluster
15     _sizes))
16   for (i in seq_along(m_values)) {
17     m <- m_values[i]
18     print(m)
19     for (j in seq_along(cluster_sizes)) {
20       cs <- cluster_sizes[j]
21
22       counts <- sample(5:15, m, replace = TRUE)
23       names(counts) <- paste0("L", seq_len(m))
24
25       clusters <- unname(split(names(counts), ceiling(seq_along(
26         counts) / cs)))
27       threshold <- (sum(counts) / m * cs) / 3
28
29       start_time <- Sys.time()
30       suppressWarnings(suppressMessages(
31         tryCatch({
32           maximum_mutual_information_hierarchical(counts, threshold
33             , clusters)
34         }, error = function(e) {
35           NULL
36         })
37       ))
38       end_time <- Sys.time()
39     }
40   }
41 }

```

```

37     data[i, j] <- as.numeric(difftime(end_time, start_time, units
    = "secs"))
38   }
39 }
40 write.csv(data, cache_path, row.names = FALSE)
41 } else {
42   data <- as.matrix(read.csv(cache_path))
43 }
44
45 dimnames(data) <- NULL
46 data <- log10(pmax(data, 0.001))
47
48 colors <- colorRampPalette(c("forestgreen", "gold", "darkorange", "
    firebrick"))(50)
49
50 pdf(output_path, width = 6, height = 4)
51
52 layout(matrix(c(1, 2), nrow = 1), widths = c(4, 1))
53
54 par(mar = c(5, 4, 4, 1))
55 image(x = m_values,
56       y = cluster_sizes,
57       z = data,
58       col = colors,
59       xlab = "Total Number of Levels (m)",
60       ylab = "Maximum Component Size (c)",
61       las = 1)
62
63 grid(nx = length(m_values), ny = length(cluster_sizes), col = "
    black", lty = "dotted")
64
65 par(mar = c(5, 0, 4, 4))
66
67 z_min <- min(data)
68 z_max <- max(data)
69
70 legend_matrix <- matrix(seq(z_min, z_max, length.out = length(
    colors)), nrow = 1)
71
72 image(x = 1,
73       y = seq(z_min, z_max, length.out = length(colors)),
74       z = legend_matrix,
75       col = colors,
76       axes = FALSE,
77       xlab = "",
78       ylab = "")
79
80 tick_values <- c(0.001, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5,
    10, 20, round(10^z_max, digits = 1))
81 axis(4, at = log10(tick_values), labels = tick_values, las = 1)
82
83 dev.off()

```

Listing A.5: hierarchical`benchmark.R

```

1 set.seed(0)
2 n <- 10000
3
4 x1 <- sample(as.factor(c("A", "B", "C", "D", "E")), n, replace =
    TRUE, prob = c(0.5, 0.1, 0.18, 0.2, 0.02))
5 x2 <- sample(as.factor(c("F", "G", "H")), n, replace = TRUE)
6 x3 <- rnorm(n)
7 x4 <- ordered(

```

```

8   sample(c("Very Low", "Low", "Medium", "High", "Very High"), n,
9         replace = TRUE, prob = c(0.05, 0.25, 0.4, 0.2, 0.1)),
10  levels = c("Very Low", "Low", "Medium", "High", "Very High")
11 )
12 x1_to_num <- ifelse(x1 == "A", 0, ifelse(x1 == "B", 2, ifelse(x1 ==
13   "C", 3, ifelse(x1 == "D", 4, 0))))
14 x_4_to_num <- ifelse(x4 == "Very Low", 5, ifelse(x4 == "Low", 3,
15   ifelse(x4 == "Medium", 1, ifelse(x4 == "High", 4, 6))))
16 y_true <- x1_to_num + 2*x3 + .5*x_4_to_num
17 y <- y_true + rnorm(n, sd = 2)
18 y_disc <- as.numeric(y < 10)
19 data <- data.frame(y_true, y, y_disc, x1, x2, x3, x4)
20
21 # shared config:
22 n_samples <- 5000
23 sample_size <- 100
24 base_font_size <- 20

```

Listing A.6: generate_data.R

```

1  library(optLump)
2  library(ggplot2)
3
4  data_path <- if (file.exists("generate_data.R")) "generate_data.R"
5    else "figures/generate_data.R"
6  source(data_path)
7
8  args <- commandArgs(trailingOnly = TRUE)
9
10 cache_path <- "figures/ci_ordinal_cache.csv"
11 output_path <- "figures/ci_ordinal_plot.pdf"
12
13 if ("--rebuild" %in% args || !file.exists(cache_path)) {
14   conf_interval <- function(q_over_n) {
15     print(q_over_n)
16     q <- q_over_n * sample_size
17     result <- replicate(n_samples, {
18       vals <- table(sample(data$x4, sample_size, replace = TRUE))
19       maximum_mutual_information_ordinal(vals, q)$mutual_
20       information
21     })
22     c(
23       lower_bound = unname(quantile(result, probs = 0.025)),
24       sample_mean = mean(result),
25       upper_bound = unname(quantile(result, probs = 0.975))
26     )
27   }
28
29   thresholds <- seq(0.01, 0.6, by = 0.01)
30   intervals <- sapply(thresholds, conf_interval)
31   plot_data <- data.frame(
32     threshold = thresholds,
33     lower_bound = intervals["lower_bound", ],
34     sample_mean = intervals["sample_mean", ],
35     upper_bound = intervals["upper_bound", ]
36   )
37   write.csv(plot_data, cache_path, row.names = FALSE)
38 } else {
39   plot_data <- read.csv(cache_path)
40 }

```

```

39 plot_data |>
40   ggplot(aes(x = threshold)) +
41     geom_ribbon(aes(ymin = lower_bound, ymax = upper_bound), fill = "
42       steelblue", alpha = 0.3) +
43     geom_line(aes(y = sample_mean), color = "steelblue", linewidth =
44       1) +
45     theme_minimal(base_size = base_font_size) +
46     labs(
47       x      = "Threshold (q/n)",
48       y      = "Mutual Information (nats)"
49     )
50 ggsave(output_path, width = 6, height = 4)

```

Listing A.7: ci ordinal plot.R

```

1 library(optLump)
2 library(ggplot2)
3
4 data_path <- if (file.exists("generate_data.R")) "generate_data.R"
5   else "figures/generate_data.R"
6 source(data_path)
7
8 args <- commandArgs(trailingOnly = TRUE)
9
10 cache_path <- "figures/ci_nominal_cache.csv"
11 output_path <- "figures/ci_nominal_plot.pdf"
12
13 if ("--rebuild" %in% args || !file.exists(cache_path)) {
14   conf_interval <- function(q_over_n) {
15     print(q_over_n)
16     q <- q_over_n * sample_size
17     result <- replicate(n_samples, {
18       vals <- table(sample(data$x1, sample_size, replace = TRUE))
19       maximum_mutual_information_nominal(vals, q)$mutual_
20       information
21     })
22     c(
23       lower_bound = unname(quantile(result, probs = 0.025)),
24       sample_mean = mean(result),
25       upper_bound = unname(quantile(result, probs = 0.975))
26     )
27   }
28
29   thresholds <- seq(0.01, 0.6, by = 0.01)
30   intervals <- sapply(thresholds, conf_interval)
31   plot_data <- data.frame(
32     threshold = thresholds,
33     lower_bound = intervals["lower_bound", ],
34     sample_mean = intervals["sample_mean", ],
35     upper_bound = intervals["upper_bound", ]
36   )
37   write.csv(plot_data, cache_path, row.names = FALSE)
38 } else {
39   plot_data <- read.csv(cache_path)
40 }
41
42 plot_data |>
43   ggplot(aes(x = threshold)) +
44     geom_ribbon(aes(ymin = lower_bound, ymax = upper_bound), fill = "
45       steelblue", alpha = 0.3) +

```

```

43 geom_line(aes(y = sample_mean), color = "steelblue", linewidth =
44   1) +
45   theme_minimal(base_size = base_font_size) +
46   labs(
47     x      = "Threshold (q/n)",
48     y      = "Mutual Information (nats)"
49   )
50 ggsave(output_path, width = 6, height = 4)

```

Listing A.8: ci'nominal'plot.R

```

1 library(optLump)
2 library(ggplot2)
3
4 data_path <- if (file.exists("generate_data.R")) "generate_data.R"
5   else "figures/generate_data.R"
6 source(data_path)
7
8 args <- commandArgs(trailingOnly = TRUE)
9
10 cache_path <- "figures/ci_heuristic_cache.csv"
11 output_path <- "figures/ci_heuristic_plot.pdf"
12
13 if ("--rebuild" %in% args || !file.exists(cache_path)) {
14   conf_interval <- function(q_over_n) {
15     print(q_over_n)
16     q <- q_over_n * sample_size
17     result <- replicate(n_samples, {
18       vals <- table(sample(data$x1, sample_size, replace = TRUE))
19       maximum_mutual_information_nominal_heuristic(vals, q)$
20       mutual_information
21     })
22     c(
23       lower_bound = unname(quantile(result, probs = 0.025)),
24       sample_mean = mean(result),
25       upper_bound = unname(quantile(result, probs = 0.975))
26     )
27   }
28
29   thresholds <- seq(0.01, 0.6, by = 0.01)
30   intervals <- sapply(thresholds, conf_interval)
31   plot_data <- data.frame(
32     threshold = thresholds,
33     lower_bound = intervals["lower_bound", ],
34     sample_mean = intervals["sample_mean", ],
35     upper_bound = intervals["upper_bound", ]
36   )
37   write.csv(plot_data, cache_path, row.names = FALSE)
38 } else {
39   plot_data <- read.csv(cache_path)
40 }
41
42 plot_data |>
43   ggplot(aes(x = threshold)) +
44   geom_ribbon(aes(ymin = lower_bound, ymax = upper_bound), fill = "
45     steelblue", alpha = 0.3) +
46   geom_line(aes(y = sample_mean), color = "steelblue", linewidth =
47     1) +
48   theme_minimal(base_size = base_font_size) +
49   labs(
50     x      = "Threshold (q/n)",
51     y      = "Mutual Information (nats)"

```

```

48 )
49
50 ggsave(output_path, width = 6, height = 4)

```

Listing A.9: ci`heuristic`plot.R

```

1  library(optLump)
2  library(ggplot2)
3
4  data_path <- if (file.exists("generate_data.R")) "generate_data.R"
5  else "figures/generate_data.R"
6  source(data_path)
7
8
9  args <- commandArgs(trailingOnly = TRUE)
10
11 cache_path <- "figures/ci_ordinal_sup_cache.csv"
12 output_path <- "figures/ci_ordinal_sup_plot.pdf"
13
14 if ("--rebuild" %in% args || !file.exists(cache_path)) {
15   conf_interval <- function(q_over_n) {
16     print(q_over_n)
17     q <- q_over_n * sample_size
18     result <- replicate(n_samples, {
19       vals <- data[sample(nrow(data), sample_size, replace = TRUE)
20     ], ]
21     vals <- table(vals$x4, vals$y_disc)
22     maximum_mutual_information_ordinal_supervised(vals, q)$
23     mutual_information
24   })
25   c(
26     lower_bound = unname(quantile(result, probs = 0.025)),
27     sample_mean = mean(result),
28     upper_bound = unname(quantile(result, probs = 0.975))
29   )
30 }
31
32 thresholds <- seq(0.01, 0.6, by = 0.01)
33 intervals <- sapply(thresholds, conf_interval)
34 plot_data <- data.frame(
35   threshold = thresholds,
36   lower_bound = intervals["lower_bound", ],
37   sample_mean = intervals["sample_mean", ],
38   upper_bound = intervals["upper_bound", ]
39 )
40
41 write.csv(plot_data, cache_path, row.names = FALSE)
42 } else {
43   plot_data <- read.csv(cache_path)
44 }
45
46 plot_data |>
47   ggplot(aes(x = threshold)) +
48   geom_ribbon(aes(ymin = lower_bound, ymax = upper_bound), fill = "
49   steelblue", alpha = 0.3) +
50   geom_line(aes(y = sample_mean), color = "steelblue", linewidth =
51   1) +
52   theme_minimal(base_size = base_font_size) +
53   labs(
54     x = "Threshold (q/n)",
55     y = "Mutual Information (nats)"
56   )

```

```
51 ggsave(output_path, width = 6, height = 4)
```

Listing A.10: ci'ordinal'sup'plot.R

```
1 library(optLump)
2 library(ggplot2)
3
4 data_path <- if (file.exists("generate_data.R")) "generate_data.R"
5   else "figures/generate_data.R"
6 source(data_path)
7
8 args <- commandArgs(trailingOnly = TRUE)
9
10 cache_path <- "figures/ci_nominal_sup_cache.csv"
11 output_path <- "figures/ci_nominal_sup_plot.pdf"
12
13 if ("--rebuild" %in% args || !file.exists(cache_path)) {
14   conf_interval <- function(q_over_n) {
15     print(q_over_n)
16     q <- q_over_n * sample_size
17     result <- replicate(n_samples, {
18       vals <- data[sample(nrow(data), sample_size, replace = TRUE)
19     ], ]
20     vals <- table(vals$x1, vals$y_disc)
21     maximum_mutual_information_nominal_supervised(vals, q)$
22     mutual_information
23   })
24   c(
25     lower_bound = unname(quantile(result, probs = 0.025)),
26     sample_mean = mean(result),
27     upper_bound = unname(quantile(result, probs = 0.975))
28   )
29 }
30
31 thresholds <- seq(0.01, 0.6, by = 0.01)
32 intervals <- sapply(thresholds, conf_interval)
33 plot_data <- data.frame(
34   threshold = thresholds,
35   lower_bound = intervals["lower_bound", ],
36   sample_mean = intervals["sample_mean", ],
37   upper_bound = intervals["upper_bound", ]
38 )
39
40 write.csv(plot_data, cache_path, row.names = FALSE)
41 } else {
42   plot_data <- read.csv(cache_path)
43 }
44
45 plot_data |>
46 ggplot(aes(x = threshold)) +
47   geom_ribbon(aes(ymin = lower_bound, ymax = upper_bound), fill = "
48     steelblue", alpha = 0.3) +
49   geom_line(aes(y = sample_mean), color = "steelblue", linewidth =
50     1) +
51   theme_minimal(base_size = base_font_size) +
52   labs(
53     x = "Threshold (q/n)",
54     y = "Mutual Information (nats)"
55   )
56
57 ggsave(output_path, width = 6, height = 4)
```

Listing A.11: ci'nominal'sup'plot.R

```

1 library(optLump)
2 library(ggplot2)
3
4 data_path <- if (file.exists("generate_data.R")) "generate_data.R"
5   else "figures/generate_data.R"
6 source(data_path)
7
8
9 args <- commandArgs(trailingOnly = TRUE)
10
11 cache_path <- "figures/heuristic_gap_cache.csv"
12 output_path <- "figures/heuristic_gap_plot.pdf"
13
14 if ("--rebuild" %in% args || !file.exists(cache_path)) {
15   conf_interval <- function(q_over_n) {
16     print(q_over_n)
17     q <- q_over_n * sample_size
18     result <- replicate(n_samples, {
19       vals <- table(sample(data$x1, sample_size, replace = TRUE))
20       optimal <- maximum_mutual_information_nominal(vals, q)$
21       mutual_information
22       smart <- maximum_mutual_information_nominal_heuristic(vals,
23         q, heuristic = "smart")$mutual_information
24       largest <- maximum_mutual_information_nominal_heuristic(
25         vals, q, heuristic = "largest")$mutual_information
26       other <- maximum_mutual_information_nominal_heuristic(vals,
27         q, heuristic = "other")$mutual_information
28
29       c(
30         smart = optimal - smart,
31         largest = optimal - largest,
32         other = optimal - other
33       )
34     })
35
36   data.frame(
37     threshold = q_over_n,
38     heuristic = c("smart", "largest", "other"),
39     lower_bound = apply(result, 1, quantile, probs = 0.025),
40     sample_mean = apply(result, 1, mean),
41     upper_bound = apply(result, 1, quantile, probs = 0.975)
42   )
43 }
44
45 thresholds <- seq(0.01, 0.6, by = 0.01)
46
47 intervals_list <- lapply(thresholds, conf_interval)
48 plot_data <- do.call(rbind, intervals_list)
49 rownames(plot_data) <- NULL
50
51 write.csv(plot_data, cache_path, row.names = FALSE)
52 } else {
53   plot_data <- read.csv(cache_path)
54 }
55
56 plot_data |>
57   ggplot(aes(x = threshold, color = heuristic, fill = heuristic)) +
58   geom_ribbon(aes(ymin = lower_bound, ymax = upper_bound), alpha =
59     0.3, color = NA) +
60   geom_line(aes(y = sample_mean), linewidth = 1) +
61   theme_minimal() +
62   labs(
63     x = "Threshold (q/n)",

```

```

57     y       = "Gap to Optimal (nats)",
58     color   = "Heuristic",
59     fill     = "Heuristic"
60   )
61
62   ggsave(output_path, width = 6, height = 4)

```

Listing A.12: heuristic'gap'plot.R

```

1  library(optLump)
2  library(ggplot2)
3
4  set.seed(0)
5  n <- 20000
6
7  m_nom <- 10
8  nom_probs <- c(0.30, 0.25, 0.21, 0.04, 0.04, 0.04, 0.03, 0.03,
9                0.03, 0.03)
9  x1 <- sample(factor(seq_len(m_nom)), n, replace = TRUE, prob = nom_
10               probs)
11 nom_effect <- c(0, 2.0, 4.0, rep(6.0, 7)) + rnorm(m_nom, sd = 0.15)
12
13 m_ord <- 18
14 ord_probs <- (1 / seq_len(m_ord))^0.85
15 ord_probs <- ord_probs / sum(ord_probs)
16 ord_levels <- as.character(seq_len(m_ord))
17 x4 <- ordered(
18   sample(ord_levels, n, replace = TRUE, prob = ord_probs),
19   levels = ord_levels
20 )
21
22 ord_effect <- c(0, 1, 2, 3, 4, 5, rep(6, m_ord - 6)) + rnorm(m_ord,
23   sd = 0.15)
24
25 x3 <- rnorm(n)
26
27 noise_sd <- 1.5
28
29 y_nom_true <- nom_effect[as.integer(x1)] + 0.5 * x3
30 y_nom <- y_nom_true + rnorm(n, sd = noise_sd)
31 y_nom_disc <- as.factor(as.numeric(y_nom > median(y_nom)))
32
33 y_ord_true <- ord_effect[as.integer(x4)] + 0.5 * x3
34 y_ord <- y_ord_true + rnorm(n, sd = noise_sd)
35 y_ord_disc <- as.factor(as.numeric(y_ord > median(y_ord)))
36
37 data <- data.frame(
38   x1, x4, x3,
39   y_nom, y_nom_true, y_nom_disc,
40   y_ord, y_ord_true, y_ord_disc
41 )
42
43 # shared config:
44 n_samples <- 1000
45 sample_size <- 80
46 n_test <- 1000
47 base_font_size <- 20
48
49 safe_predict <- function(model, newdata) {
50   for (v in names(model$xlevels)) {
51     if (v %in% names(newdata)) {

```

```

52     bad <- !(as.character(newdata[[v]]) %in% model$xlevels[[v]])
53     if (any(bad, na.rm = TRUE)) newdata[[v]][bad] <- NA
54   }
55 }
56 predict(model, newdata)
57 }
58
59 to_long <- function(d) {
60   long <- rbind(
61     data.frame(threshold = d$threshold, metric = "Variance",
62               model = "Lumped", value = d$var_lumped),
63     data.frame(threshold = d$threshold, metric = "Bias2", model = "Lumped",
64               value = d$bias_lumped),
65     data.frame(threshold = d$threshold, metric = "MSE",
66               model = "Lumped", value = d$mse_lumped),
67     data.frame(threshold = d$threshold, metric = "Variance",
68               model = "Unlumped", value = d$var_unlumped),
69     data.frame(threshold = d$threshold, metric = "Bias2", model = "Unlumped",
70               value = d$bias_unlumped),
71     data.frame(threshold = d$threshold, metric = "MSE",
72               model = "Unlumped", value = d$mse_unlumped)
73   )
74   long$metric <- factor(long$metric, levels = c("MSE", "Bias2", "Variance"))
75   long
76 }

```

Listing A.13: generate_data_mse.R

```

1  data_path <- if (file.exists("generate_data_mse.R")) "generate_data
   _mse.R" else "figures/generate_data_mse.R"
2  source(data_path)
3
4  args <- commandArgs(trailingOnly = TRUE)
5
6  cache_path <- "figures/mse_ordinal_cache.csv"
7  output_path <- "figures/mse_ordinal.pdf"
8
9
10 if ("--rebuild" %in% args || !file.exists(cache_path)) {
11   test_data <- data[sample(nrow(data), n_test, replace = TRUE), ]
12
13   performance <- function(q_over_n) {
14     print(q_over_n)
15     q <- q_over_n * sample_size
16
17     result <- replicate(n_samples, {
18       s <- data[sample(nrow(data), sample_size, replace = TRUE), ]
19
20       s$xL <- lump_ordinal(s$x4, q)
21       map <- setNames(as.character(s$xL), as.character(s$x4))
22       map <- map[!duplicated(names(map))]
23
24       test_lp <- test_data
25       test_lp$xL <- ordered(map[as.character(test_data$x4)], levels
   = levels(s$xL))
26
27       m_un <- lm(y_ord ~ x4 + x3, data = s)
28       m_lp <- if (nlevels(droplevels(s$xL)) > 1) {
29         lm(y_ord ~ xL + x3, data = s)
30       } else {
31         lm(y_ord ~ x3, data = s)

```

```

32     }
33
34     list(
35       un = safe_predict(m_un, test_data),
36       lp = safe_predict(m_lp, test_lp)
37     )
38   }, simplify = FALSE)
39
40   decomp <- function(key) {
41     preds <- do.call(cbind, lapply(result, '[', key))
42     mean_pred <- rowMeans(preds, na.rm = TRUE)
43     bias2 <- mean((mean_pred - test_data$y_ord_true)^2, na.rm =
44 TRUE)
45     variance <- mean(apply(preds, 1, var, na.rm = TRUE), na.rm =
46 TRUE)
47     c(variance = variance, bias2 = bias2, mse = bias2 + variance)
48   }
49
50   c(un = decomp("un"), lp = decomp("lp"))
51 }
52
53 thresholds <- seq(0.01, 0.25, by = 0.012)
54 result <- sapply(thresholds, performance)
55 plot_data <- data.frame(
56   threshold      = thresholds,
57   var_lumped     = result["lp.variance", ],
58   bias_lumped    = result["lp.bias2", ],
59   mse_lumped     = result["lp.mse", ],
60   var_unlumped   = result["un.variance", ],
61   bias_unlumped  = result["un.bias2", ],
62   mse_unlumped   = result["un.mse", ]
63 )
64 write.csv(plot_data, cache_path, row.names = FALSE)
65 } else {
66   plot_data <- read.csv(cache_path)
67 }
68
69 to_long(plot_data) |>
70 ggplot(aes(x = threshold, y = value, color = metric, linetype =
71 model)) +
72 geom_line(linewidth = 1) +
73 scale_linetype_manual(values = c(Lumped = "solid", Unlumped = "
74 longdash")) +
75 theme_minimal(base_size = base_font_size) +
76 theme(
77   legend.position = "bottom",
78   legend.box = "vertical",
79   legend.margin = margin(0, 0, 0, 0),
80   legend.spacing.y = unit(0, "pt"),
81   legend.text = element_text(size = 13),
82   legend.title = element_text(size = 14)
83 ) +
84 guides(
85   color = guide_legend(keywidth = unit(0.9, "cm")),
86   linetype = guide_legend(keywidth = unit(1.8, "cm"))
87 ) +
88 labs(
89   x = "Threshold (q/n)",
90   y = "Error",
91   color = "Quantity",
92   linetype = "Model"

```

```

90 ) +
91 coord_cartesian(ylim = c(0, 1.25))
92
93 ggsave(output_path, width = 6, height = 4)

```

Listing A.14: mse.ordinal.R

```

1 data_path <- if (file.exists("generate_data_mse.R")) "generate_data
  _mse.R" else "figures/generate_data_mse.R"
2 source(data_path)
3
4 args <- commandArgs(trailingOnly = TRUE)
5
6 cache_path <- "figures/mse_nominal_cache.csv"
7 output_path <- "figures/mse_nominal.pdf"
8
9
10 if ("--rebuild" %in% args || !file.exists(cache_path)) {
11   test_data <- data[sample(nrow(data), n_test, replace = TRUE), ]
12
13   performance <- function(q_over_n) {
14     print(q_over_n)
15     q <- q_over_n * sample_size
16
17     result <- replicate(n_samples, {
18       s <- data[sample(nrow(data), sample_size, replace = TRUE), ]
19
20       s$xL <- lump_nominal(s$x1, q)
21       map <- setNames(as.character(s$xL), as.character(s$x1))
22       map <- map[!duplicated(names(map))]
23
24       test_lp <- test_data
25       test_lp$xL <- factor(map[as.character(test_data$x1)], levels
= levels(s$xL))
26
27       m_un <- lm(y_nom ~ x1 + x3, data = s)
28       m_lp <- if (nlevels(droplevels(s$xL)) > 1) {
29         lm(y_nom ~ xL + x3, data = s)
30       } else {
31         lm(y_nom ~ x3, data = s)
32       }
33
34       list(
35         un = safe_predict(m_un, test_data),
36         lp = safe_predict(m_lp, test_lp)
37       )
38     }, simplify = FALSE)
39
40     decomp <- function(key) {
41       preds <- do.call(cbind, lapply(result, '[', key))
42       mean_pred <- rowMeans(preds, na.rm = TRUE)
43       bias2 <- mean((mean_pred - test_data$y_nom_true)^2, na.rm =
TRUE)
44       variance <- mean(apply(preds, 1, var, na.rm = TRUE), na.rm =
TRUE)
45       c(variance = variance, bias2 = bias2, mse = bias2 + variance)
46     }
47
48     c(un = decomp("un"), lp = decomp("lp"))
49   }
50
51 thresholds <- seq(0.01, 0.25, by = 0.012)
52 result <- sapply(thresholds, performance)

```

```

53 plot_data <- data.frame(
54   threshold      = thresholds,
55   var_lumped     = result["lp.variance", ],
56   bias_lumped    = result["lp.bias2", ],
57   mse_lumped     = result["lp.mse", ],
58   var_unlumped   = result["un.variance", ],
59   bias_unlumped  = result["un.bias2", ],
60   mse_unlumped   = result["un.mse", ]
61 )
62 write.csv(plot_data, cache_path, row.names = FALSE)
63 } else {
64   plot_data <- read.csv(cache_path)
65 }
66
67
68 to_long(plot_data) |>
69   ggplot(aes(x = threshold, y = value, color = metric, linetype =
70     model)) +
71   geom_line(linewidth = 1) +
72   scale_linetype_manual(values = c(Lumped = "solid", Unlumped = "
73     longdash")) +
74   theme_minimal(base_size = base_font_size) +
75   theme(
76     legend.position = "bottom",
77     legend.box = "vertical",
78     legend.margin = margin(0, 0, 0, 0),
79     legend.spacing.y = unit(0, "pt"),
80     legend.text = element_text(size = 13),
81     legend.title = element_text(size = 14)
82   ) +
83   guides(
84     color = guide_legend(keywidth = unit(0.9, "cm")),
85     linetype = guide_legend(keywidth = unit(1.8, "cm"))
86   ) +
87   labs(
88     x = "Threshold (q/n)",
89     y = "Error",
90     color = "Quantity",
91     linetype = "Model"
92   ) +
93   coord_cartesian(ylim = c(0, 1.25))
94
95 ggsave(output_path, width = 6, height = 4)

```

Listing A.15: mse'nominal.R

```

1 data_path <- if (file.exists("generate_data_mse.R")) "generate_data
2   _mse.R" else "figures/generate_data_mse.R"
3 source(data_path)
4
5 args <- commandArgs(trailingOnly = TRUE)
6
7 cache_path <- "figures/mse_ordinal_sup_cache.csv"
8 output_path <- "figures/mse_ordinal_sup.pdf"
9
10 if ("--rebuild" %in% args || !file.exists(cache_path)) {
11   test_data <- data[sample(nrow(data), n_test, replace = TRUE), ]
12
13   performance <- function(q_over_n) {
14     print(q_over_n)
15     q <- q_over_n * sample_size
16
17     result <- replicate(n_samples, {

```

```

17   s <- data[sample(nrow(data), sample_size, replace = TRUE), ]
18
19   s$xL <- lump_ordinal_supervised(s$x4, s$y_ord_disc, q)
20   map <- setNames(as.character(s$xL), as.character(s$x4))
21   map <- map[!duplicated(names(map))]
22
23   test_lp <- test_data
24   test_lp$xL <- ordered(map[as.character(test_data$x4)], levels
= levels(s$xL))
25
26   m_un <- lm(y_ord ~ x4 + x3, data = s)
27   m_lp <- if (nlevels(droplevels(s$xL)) > 1) {
28     lm(y_ord ~ xL + x3, data = s)
29   } else {
30     lm(y_ord ~ x3, data = s)
31   }
32
33   list(
34     un = safe_predict(m_un, test_data),
35     lp = safe_predict(m_lp, test_lp)
36   )
37 }, simplify = FALSE)
38
39 decomp <- function(key) {
40   preds <- do.call(cbind, lapply(result, '[', key))
41   mean_pred <- rowMeans(preds, na.rm = TRUE)
42   bias2 <- mean((mean_pred - test_data$y_ord_true)^2, na.rm =
TRUE)
43   variance <- mean(apply(preds, 1, var, na.rm = TRUE), na.rm =
TRUE)
44   c(variance = variance, bias2 = bias2, mse = bias2 + variance)
45 }
46
47 c(un = decomp("un"), lp = decomp("lp"))
48 }
49
50 thresholds <- seq(0.01, 0.25, by = 0.012)
51 result <- sapply(thresholds, performance)
52 plot_data <- data.frame(
53   threshold      = thresholds,
54   var_lumped     = result["lp.variance", ],
55   bias_lumped    = result["lp.bias2", ],
56   mse_lumped     = result["lp.mse", ],
57   var_unlumped   = result["un.variance", ],
58   bias_unlumped  = result["un.bias2", ],
59   mse_unlumped   = result["un.mse", ]
60 )
61 write.csv(plot_data, cache_path, row.names = FALSE)
62 } else {
63   plot_data <- read.csv(cache_path)
64 }
65
66 to_long(plot_data) |>
67   ggplot(aes(x = threshold, y = value, color = metric, linetype =
model)) +
68   geom_line(linewidth = 1) +
69   scale_linetype_manual(values = c(Lumped = "solid", Unlumped = "
longdash")) +
70   theme_minimal(base_size = base_font_size) +
71   theme(
72     legend.position = "bottom",
73     legend.box = "vertical",

```

```

74   legend.margin = margin(0, 0, 0, 0),
75   legend.spacing.y = unit(0, "pt"),
76   legend.text = element_text(size = 13),
77   legend.title = element_text(size = 14)
78 ) +
79 guides(
80   color = guide_legend(keywidth = unit(0.9, "cm")),
81   linetype = guide_legend(keywidth = unit(1.8, "cm"))
82 ) +
83 labs(
84   x = "Threshold (q/n)",
85   y = "Error",
86   color = "Quantity",
87   linetype = "Model"
88 )
89
90 to_long(plot_data) |>
91 ggplot(aes(x = threshold, y = value, color = metric, linetype =
92   model)) +
93   geom_line(linewidth = 1) +
94   scale_linetype_manual(values = c(Lumped = "solid", Unlumped = "
95     longdash")) +
96   theme_minimal(base_size = base_font_size) +
97   theme(
98     legend.position = "bottom",
99     legend.box = "vertical",
100     legend.margin = margin(0, 0, 0, 0),
101     legend.spacing.y = unit(0, "pt"),
102     legend.text = element_text(size = 13),
103     legend.title = element_text(size = 14)
104 ) +
105 guides(
106   color = guide_legend(keywidth = unit(0.9, "cm")),
107   linetype = guide_legend(keywidth = unit(1.8, "cm"))
108 ) +
109 labs(
110   x = "Threshold (q/n)",
111   y = "Error",
112   color = "Quantity",
113   linetype = "Model"
114 ) +
115 coord_cartesian(ylim = c(0, 1.25))
116
117 ggsave(output_path, width = 6, height = 4)

```

Listing A.16: mse`ordinal`sup.R

```

1 data_path <- if (file.exists("generate_data_mse.R")) "generate_data
2   _mse.R" else "figures/generate_data_mse.R"
3 source(data_path)
4
5 args <- commandArgs(trailingOnly = TRUE)
6
7 cache_path <- "figures/mse_nominal_sup_cache.csv"
8 output_path <- "figures/mse_nominal_sup.pdf"
9
10 if ("--rebuild" %in% args || !file.exists(cache_path)) {
11   test_data <- data[sample(nrow(data), n_test, replace = TRUE), ]
12
13   performance <- function(q_over_n) {
14     print(q_over_n)
15     q <- q_over_n * sample_size

```

```

16 result <- replicate(n_samples, {
17   s <- data[sample(nrow(data), sample_size, replace = TRUE), ]
18
19   s$xL <- lump_nominal_supervised(s$x1, s$y_nom_disc, q)
20   map <- setNames(as.character(s$xL), as.character(s$x1))
21   map <- map[!duplicated(names(map))]
22
23   test_lp <- test_data
24   test_lp$xL <- factor(map[as.character(test_data$x1)], levels
25 = levels(s$xL))
26
27   m_un <- lm(y_nom ~ x1 + x3, data = s)
28   m_lp <- if (nlevels(droplevels(s$xL)) > 1) {
29     lm(y_nom ~ xL + x3, data = s)
30   } else {
31     lm(y_nom ~ x3, data = s)
32   }
33
34   list(
35     un = safe_predict(m_un, test_data),
36     lp = safe_predict(m_lp, test_lp)
37   )
38 }, simplify = FALSE)
39
40 decomp <- function(key) {
41   preds <- do.call(cbind, lapply(result, '[', key))
42   mean_pred <- rowMeans(preds, na.rm = TRUE)
43   bias2 <- mean((mean_pred - test_data$y_nom_true)^2, na.rm =
44 TRUE)
45   variance <- mean(apply(preds, 1, var, na.rm = TRUE), na.rm =
46 TRUE)
47   c(variance = variance, bias2 = bias2, mse = bias2 + variance)
48 }
49
50 c(un = decomp("un"), lp = decomp("lp"))
51 }
52
53 thresholds <- seq(0.01, 0.25, by = 0.012)
54 result <- sapply(thresholds, performance)
55 plot_data <- data.frame(
56   threshold = thresholds,
57   var_lumped = result["lp.variance", ],
58   bias_lumped = result["lp.bias2", ],
59   mse_lumped = result["lp.mse", ],
60   var_unlumped = result["un.variance", ],
61   bias_unlumped = result["un.bias2", ],
62   mse_unlumped = result["un.mse", ]
63 )
64 write.csv(plot_data, cache_path, row.names = FALSE)
65 } else {
66   plot_data <- read.csv(cache_path)
67 }
68
69 to_long(plot_data) |>
70 ggplot(aes(x = threshold, y = value, color = metric, linetype =
71 model)) +
72 geom_line(linewidth = 1) +
73 scale_linetype_manual(values = c(Lumped = "solid", Unlumped = "
74 longdash")) +
75 theme_minimal(base_size = base_font_size) +
76 theme(

```

```

73     legend.position = "bottom",
74     legend.box = "vertical",
75     legend.margin = margin(0, 0, 0, 0),
76     legend.spacing.y = unit(0, "pt"),
77     legend.text = element_text(size = 13),
78     legend.title = element_text(size = 14)
79   ) +
80   guides(
81     color = guide_legend(keywidth = unit(0.9, "cm")),
82     linetype = guide_legend(keywidth = unit(1.8, "cm"))
83   ) +
84   labs(
85     x = "Threshold (q/n)",
86     y = "Error",
87     color = "Quantity",
88     linetype = "Model"
89   ) +
90   coord_cartesian(ylim = c(0, 1.25))
91
92   ggsave(output_path, width = 6, height = 4)

```

Listing A.17: mse'nominal'sup.R