TECHNICAL UNIVERSITY OF MUNICH

TUM Data Innovation Lab

"Scalable Statistics with Large Datasets"

Authors	Cheng-wei Wang, Martin Rosenzweig, Nino Mumladze, Tatiana Kuznetsova, Thomas Decker
Mentors	Aurélien Ouattara, Florian Felice, Liubomyr Bregman Amazon
Co-mentor	M.Sc Özge Sahin (Department of Mathematics)
Project Lead	Dr. Ricardo Acevedo Cabra (Department of Mathematics)
Supervisor	Prof. Dr. Massimo Fornasier (Department of Mathematics)

Jul 2020

Contents

1 Introduction

In the era of "big data", the decision-making process for businesses has taken a different form and involves new approaches. The numbers are now considered to "speak for themselves", and strategic actions are more often taken and rationalized by data. Statistics has become a crucial tool for business owners to make informed decisions, draw valid conclusions, and act based on them. In the context of a lack of knowledge and certainty for a particular problem, utilizing big data can substantially reduce the level of uncertainty.

Recently, there has been a significant burst in the speed at which software tools are created. These are targeted to help many institutions to operate better and develop more automated ways of tackling business problems (Apache Spark, 2014 [31], TensorFlow, 2015 [1], PyTorch, 2016 [25]). These technologies are mostly designed for solving machine learning problems and are therefore particularly suited and well-optimized to perform predictive modeling. However, despite this large amount of data at our disposal and the increasing number of tools dedicated to handling large datasets, performing explanatory statistical analysis using big data volume is still an open problem in the econometrics and statistics community, due to the complexity of computation and scalability issues [10].

The issues of classical econometrics studies lie in a lack of unified tools that perform well both on modeling and retrieving statistical inference with extra-large datasets. Specifically, the tools used within Amazon Operations provide quick access to such data. Although those existing technologies are suited for operating on big data and producing predictive models [1], they lack statistics-specific outputs and are therefore not optimized for econometric evaluations. On the other hand, the performance of established solutions for econometrics such as R, SPSS, SAS run solely on in-memory data and do not exploit parallelism within their language primitives [10]. Therefore, these are often no longer efficient on large datasets.

One of our primary goals in this project is to utilize existing big data technologies thoroughly and to extend their APIs by developing econometric tools that will perform efficiently at scale and allow for causality inference.

The context of our project is a specific business problem, where we investigate a marginal effect of a particular independent variable X on a binary response Y, controlling for a set of independent variables Z. We support our analysis using a highly-imbalanced ($80\% - 20\%$ distribution in the response) dataset with more than 500 million observations and approximately 40 features. We work with panel data, which involves repeated observations on the same units over time.

There are many ways to tackle the aforementioned business problem. Our objective is to find the best specified and most efficient model. Our first approach is to disregard the differences in observations over time and to fit pooled logistic and probit regressions. However, fitting such models and especially causal inference gets challenging for extra-large datasets. Hence, we develop a novel way to efficiently compute the Fisher information over large datasets in a distributed way, using multiple graphics processing units (GPUs) with

TensorFlow. Even though this approach can alleviate the scalability issue and produce the estimates, these do not capture important unobserved group characteristics. Therefore, our exploration leads us to develop unobserved effects models for logistic regression, using conditional and unconditional maximum likelihood estimations. Those approaches make it possible to account for group-specific heterogeneity in the panel data. For all the frameworks mentioned above, we compute inference statistics, average partial effects, and resolve the question with TensorFlow and Apache Spark.

We start by introducing an overview of the existing relevant big data technologies along with packages in R. Chapter 3 provides details on the dataset and reasoning behind the methodology choice. Chapters 4 and 5 present the models' setup and further details on the developed functionalities for pooled regression and fixed-effects logit, respectively. Since we use real data collected by Amazon, we disclose only general information regarding the features and also change their naming conventions in all corresponding parts of the report.

2 Selected Technologies

When dealing with extra-large datasets, it is crucial to give a great deal of attention to performance and scalability. Since our problem is to create a robust and scalable statistical package for causal inference, we decide to implement it with TensorFlow and Apache Spark. In this section, we provide an overview of existing relevant big data technologies along with statistical packages in R, which are widely used for our purposes in econometrics.

2.1 Big data tools

TensorFlow is an open-source machine learning platform, initially developed by Google for large numerical computations and extensively used for deep learning applications. Calculations are represented in the form of computational graphs, where each node denotes an operation, and each edge indicates a multidimensional array (tensor) or a control dependency between two nodes. Once the graph is created, the execution can be seamlessly distributed across several GPUs [1].

Keras is a user-friendly modular high-level API within TensorFlow for building and training neural networks. We use it to model pooled logistic and probit regressions as a single-layer neural network with corresponding activation functions as shown in Figure 2 and is discussed in the subsequent question. TensorFlow also supplies an API for building efficient input data pipelines for handling, reading, and performing complex transformations over large amounts of data. As it is practically impossible to fit our extra-large dataset in RAM, we utilize this functionality to parallelize the data loading, preprocessing, and fitting in a batched manner across multiple central processing units (CPUs).

It is worth noticing that all the parts of the project related to TensorFlow are being performed on an ml.p2.8xlarge instance on Amazon SageMaker with 32 virtual CPUs (memory: 488 GiB) and 8 Tesla K80 GPUs (memory: 96 GPU Mem (GiB)). Amazon

SageMaker is a cloud platform that allows for building, training, and deploying machine learning models within instance-based notebooks.

Apache Spark is a distributed general-purpose cluster-computing tool that offers multiple, high and low-level, easy to use API-s, that conveniently encapsulate dealing with distributed computing and big data processing. Apache Spark's API, which is supported in many popular languages used for machine learning (Python, Scala, R..), is a flexible tool for leveraging computation capabilities of many computer clusters. The main feature of Apache Spark is operating in-memory, which ensures efficient computation. Therefore, a significant aspect of performance in Apache Spark applications is driven by the available main memory on the machine that the code is executed on. In our setting, the types of machines we use for performing a task are configurable, which allows us to be in control of runtime speed, as well as the cost of the execution.

2.2 Existing packages

Before implementing the solutions in our selected tools, i.e., Apache Spark and TensorFlow, we, firstly, review the available packages in classical statistical software. We choose the open-source and platform-independent software, "R", one of the most prominent programming languages in statistics and econometrics, which provides free statistical software environments [11]. For our analysis, we use several of those as a reference to benchmark our implementations across different frameworks.

One of the earliest implementations of generalized linear models without regularization is available in the "stats" library under glm() function. Since it allows for arbitrary link function that relates the mean of the dependent variable to the predictors, we utilize glm() to benchmark both logit and probit models. We investigate its performance on small datasets and compare the outputs with our solution in TensorFlow. As this library is not optimized for estimating the models on extra-large datasets, it makes it infeasible to benchmark on those.

In order to account for unobserved individual characteristics with our panel data, we develop unobserved effects methodologies. Before implementing our own package for unobserved effects, we explore the existing implementations that solve logistic regression with unobserved effects, to review and benchmark the computational challenges that we should address in our solution. There are two main packages that implement fixed effects in R. One is a survival package with function clogit, which maximizes the conditional likelihood. Another one is glmmML or bife, which, in contrast, uses an unconditional approach. We dive deeply into implementation and methodologies inside clogit and bife. Let us take a deeper look at "clogit package" and the related "survival" package.

2.2.1 Introduction to clogit package in R

clogit fits a regression model that has different names in various fields. In biostatistics and epidemiology, it is referred to as conditional logistic regression for matched case-control

groups, while in economics and other social studies, it is often referred to as a fixed-effects logit for panel data. It estimates the model by maximizing the exact conditional likelihood and also supports some approximation of the likelihood function.

Technically, the log likelihood for a conditional logistic regression model is equivalent to log likelihood from a Cox model with a particular data structure. In detail, a stratified Cox model with each case/control group assigned to its own stratum, time set to a constant, status of 1=case 0=control, and using the exact partial likelihood has the same likelihood formula as a conditional logistic regression. This is why an object of class "clogit" is just a wrapper for a "coxph" object. The conditional logistic regression model is available in coxph through the ties='exact' option.

It is worth mentioning that the odds ratios for events are calculated using the same partial likelihood from the Cox Model. Using predictions from the estimated odds ratios, the ranking is updated to account for what is now known about these matched sets' risk due to unmeasured factors (since our updated predictions take better account of the measured risk factors using odds ratios). This process iterates until there is an agreement (or convergence) using an expectation maximization framework. This is why clogit takes so much longer to converge than a simple Cox Model.

2.2.2 Inside R survival package

The function of interest, coxph, belongs to survival package in R with source code here. The main function coxph.R calls the wrapper (interface) coxph.fit.R, which serves as an indicator for fitting the model approximately. On the other hand, inside the coxph.R, a wrapper function coxeact.fit is called as an indicator for fitting the model exactly. Inside of both wrapping functions, a .Call() function call is initiated, calling a function in source code written in C and compiles via R CMD SHLI, creating a dll to be loaded into R. Coxfit6.c and coxecact.c are the ones that actually do the work. Specifically, for updating the parameter β, it uses the Newton method. For inverting the information matrix, it exploits the symmetry of information matrix by Cholesky decomposition and calls chsolve2 to solve linear equations, $Ax = y$ and replace y with x.

The good news in terms of computational efficiency is that the survival package allows for an exact likelihood computation or a Breslow or Efron approximation. The Breslow approximation used to be the default for most software programs, because of computational simplicity. But it is no longer the case in R.The Efron approximation is the default in R coxph().

3 Data Exploration and Methodology Overview

Having introduced the technologies that support our analysis, in the succeeding subsections we provide essential characteristics of the dataset along with the reasoning behind the choice of the models.

3.1 Data overview

As in any data science project, our project heavily relies on data preprocessing, such as data cleaning and outlier detection. This process requires learning about the business domain we are operating in as well as the necessary terminology to interpret the given data points. Afterwards, we begin a process of removing constant variables, plotting the outlier distributions, and cleaning the missing values, which mostly involves equating them to zero. Preparing the data to be analyzed also involves transforming the data format to be "wide", i.e., encoding the categorical variables to be represented as dummy binary variables.

The dataset consists of around 40 features and more than $n = 10^8$ observations. Notably, we find that the data is highly imbalanced, with about 85% of the dependent variables equal to zero. Furthermore, we observe that approximately 30 of our independent variables are categorical, which implies high sparsity of the data.

We also conduct additional high-level descriptive analyses to improve our understanding of business context and relationships between features and the dependent variable.

3.2 Selected algorithms and methodology

Before diving into the algorithms for our analysis, it is important to acknowledge the type of dataset we are dealing with. The data is cross-sectional (involving many subjects at any point in time) and multi-dimensional, meaning it involves multiple measurements for each subject over time. This type of data provides a benefit of capturing behavioral changes over time, however, this comes with the cost of added complexity during the training.

In a practical scenario, it is intractable to measure all of the variables that have a causal impact on the dependent variable. Some variables are hard and expensive to measure or represent specific individual characteristics that are impossible to quantify. These are known as "unobserved effects", and there are two ways to categorize them.

Unobserved effects are characteristic of the group or individual and they do not change over time. There are two types of unobserved effects: fixed effect α_i and "random" effects. The difference between fixed and random effect is that the latter poses one additional assumption that α_i is uncorrelated with X_{it}. In the traditional approach to panel data models, an effect is called a "random" when it is treated as a random variable and "fixed effect" when it is treated as a parameter to be estimated for each cross section observation [30].
When researching panel data, there are two ways to design the analyzing process. The first one is "between estimation", in which we compare the outcome of a group to the outcome of another one. The second one is "within estimation", where we follow the change of the outcome, when the same group is changed from the control to the treatment condition.
For causal inference, within estimation offers a better analysis, since oftentimes the assumption of unit homogeneity does not hold.

We decide to explore three methods when evaluating causal impact. Firstly, pooled logistic and probit regressions and, secondly, logistic regression with fixed effects.
The dataset is not only complex in its structure, but as aforementioned, its volume also poses a challenge. In order to to analyze it entirely, the statistical packages used should be optimized for large scale computation and run in a distributed manner.

4 Fitting our Model with Pooled Regressions

4.1 Specification of the models

In our project, we evaluate the marginal impact of an independent variable of interest X on a binary outcome Y and assuming independence and identical distribution of all observations. In the following sections, we compare the different modeling approaches we use.

Our first approach is to estimate the impact by discarding any potential unobserved heterogeneity and applying the pooled logistic regression:

$$\mathbb{P}(y_{it} = 1|x_{it}, \boldsymbol{\beta}) = \sigma(x_{it}^T\beta + \beta_0)$$

where σ is the cumulative distribution function (CDF) of a standard logistic distribution, also known as a sigmoid function.

Another way to model a binary dependent variable is given by a probit approach, which specifies the conditional probability of success given the predictors X in the following way:

$$\mathbb{P}(y_{it} = 1|x_{it}, \boldsymbol{\beta}) = \Phi(x_{it}^T\beta + \beta_0)$$

where Φ defines the CDF of a standard normal distribution. Similar to a logistic regression, this can be interpreted as modeling a real valued continuous latent variable Z with the Gaussian noise, which sign determines the dependent variable Y. More precisely, for a probit model we suppose that:

$$Z = X^T\beta + \beta_0 + \epsilon \qquad \text{and} \qquad Y = \begin{cases} 1 & \text{for} \quad Z > 0 \\ 0 & \text{for} \quad Z \leq 0 \end{cases} \qquad \text{with} \qquad \epsilon \sim \mathcal{N}(0, 1)$$

In general, logit and probit models tend to deliver very similar results in terms of estimated coefficents and partial effects in the vast majority of cases. They only are statistically distinguishable for an extremely high number of observations [9] or if many observation are located in the tails [3]. In fact, the estimated coefficients of both approaches can be used to approximate each other. From a theoretical perspective, scaling a probit estimate by $\pi/\sqrt{3} \approx 1.81$ should approximately yield the respective estimators of the logistic regression parameters since this would correct for the higher variance of the logistic distribution [2] - although empirical results imply that using a factor of 1.6 tends to be more accurate [3].

Overall a probit model is particularly reasonable if the latent variable can be expected to follow a distribution with approximately subgaussian tails whereas logistic regression is a better choice for heavier tails.

4.2 Average partial effect and partial effect at mean

In order to assess the causal inference and marginal impact of a change in a variable X on the binary response Y as well as to compare the different methods we use, we calculate average partial effects (APEs) and partial effects (PEs) at mean for all covariates. The APEs are estimated as the average of the individual marginal effects and the PEs at mean are computed for an average observation $\bar{\boldsymbol{x}}$ in the data. For logit and probit models, the corresponding calculations for continuous values are performed as follows:

$$APE_j = \beta_j \frac{\sum f(\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{\beta})}{N} \qquad PE_{atMean j} = \beta_j f(\bar{\boldsymbol{x}}^{\boldsymbol{T}}\boldsymbol{\beta})$$

where β_j is j^{th} element of interest of vector $\boldsymbol{\beta}$ (or a j^{th} independent variable), N is the number of observations, f is a probability density function (PDF) of standard logistic distribution $f(x) = \frac{e^{-x}}{(1+e^{-x})^2}$ in the case of logit and the PDF of standard normal for probit regression.

The partial effects for a discrete independent variable for both regressions are calculated as follows:

$$PE_{discrete j} = F(\beta_0 + x_{1t}\beta_1 + \beta_j(k+1)) - F(\beta_0 + x_{1t}\beta_1 + \beta_j(k))$$

where F is a corresponding CDF and k expresses a comparison to the base category, e.g. 0 for dummy variables. Example of performed calculations for pooled logistic regression is displayed in the Figure 1.

Coefficient	APE_logit	PE_at_mean_logit	APE_probit	PE_at_mean_probit
0	1.553693e-02	1.463308e-02	1.421828e-02	1.413502e-02
1	-5.740340e-06	-5.406398e-06	-6.399564e-06	-6.362088e-06
2	-5.468726e-06	-5.468726e-06	-6.437302e-06	-6.437302e-06
3	-1.019388e-03	-1.019388e-03	7.870793e-05	7.870793e-05
4	4.913330e-03	4.913330e-03	4.905477e-03	4.905477e-03

Figure 1: Partial Effects for Pooled Logistic and Probit Regressions.

Although the two approaches produce almost identical results, the main difference lies in the calculation methodology. The PEs are estimated for the average observation in the sample, however, there might not be in reality a person, company, etc. with such characteristics in data. Therefore, in practice, the calculation of the APEs is a better approach to estimate partial effects. On the other hand, the calculation of the PEs is computationally much faster, as the main overhead only lies in retrieving the mean values

for each data column. Using cloud-based data warehouses and query services, such as Amazon Redshift and Amazon Athena, this can be easily done for extra-large datasets at substantially low costs. Therefore, PEs can always be an efficient alternative for the quick estimation of the partial effects.

For completeness, it is worth noting that in the case of linear regression, both methods yield equivalent results due to the linear relationship between the independent variables and response.

4.3 Pooled logistic and probit regressions as a neural network

Our approach for estimating pooled logistic and probit regressions in distributed manner in TensorFlow is to model them as a single-neuron neural networks, where the activation function is a CDF of a standard logistic or normal distributions (Figure 2).

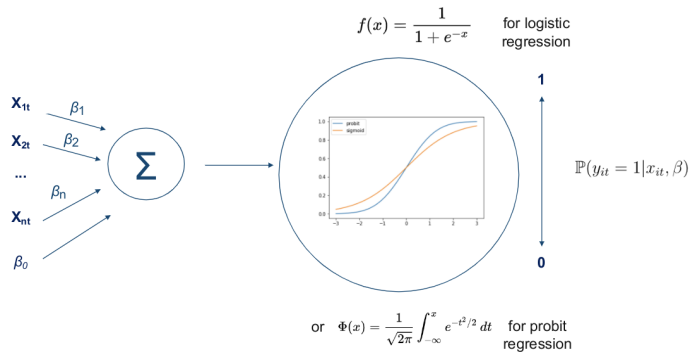

Figure 2: Logistic and Probit Regressions as a Neural Network.

To implement the probit case we use the CDF included in the TensorFlow probability API and define it as a customized activation function for the Keras model layer. As a first step and to fully utilize existing TensorFlow capabilities we are fitting the model using mini-batch gradient descent methods based on the first order approximation of the likelihood function.

This is a fundamental difference compared to the standard Fisher Scoring procedure, which is used by R and many other statistical software packages to fit generalized linear models. Such methods approximate the likelihood function by the second order Taylor series expansion and require the full batch update. Although the classical second order algorithms are well-known for their significantly faster convergence, they imply saving the whole data in memory, which is computationally infeasible when it comes to the size of our problem. This, together with very expensive data access, is one of the main reasons behind opting for the first order optimization algorithms when modeling pooled logit and probit as neural networks.

One of the mini-batch gradient-based methods we use is a stochastic gradient descent (SGD). Given a large train set with n training samples $\{x_i, y_i\}$ and a loss function

$L = \frac{1}{n}\sum_{i=1}^{n} L_i(\beta, x_i, y_i)$, we search for the best model parameters $\beta = argmin(L)$. From the trainset we choose a subset (minibatch) $B_i = \{\{x_1, y_1\}, \{x_2, y_2\}, .., \{x_m, y_m\}\}$ where $m << n$. The single update step for multiple samples is defined as follows:
$\beta^{(k+1)} = \beta^{(k)} - \alpha\nabla_\beta L(\beta^{(k)}, x_{\{1..m\}}, y_{\{1..m\}})$, with $\nabla_\beta L = \frac{1}{m}\sum_{i=1}^{m} \nabla_\beta L_i$ being a gradient for the k-th mini-batch, m is the number of observations in the current mini-batch, k refers to k-th iteration, and α is the step size, also known as learning rate. Starting trainable parameters $\beta^{(0)}$ can be initialized with a Xavier initialization [17]. As we solve a binary model, we want to minimize a strictly convex cross-entropy cost function, which is mathematically equivalent to maximizing the likelihood function. Therefore, the strict convexity together with $\nabla_\beta L$ being an unbiased estimate of the function being minimized, and non-negativity of the update parameters α ensure convergence to the global minimum.

This method imposes additional issues and constraints to the optimization process. At first, the saturating nature of the sigmoid and standard normal CDFs causes the optimization to be very sensitive to feature scaling. More precisely, if the linear combination of the features inside the activation function becomes too high in absolute value, the respective gradient will be very close to zero causing ineffective parameter updates. Thus, it is crucial to scale the feature in order to ensure consistently non zero gradients suitable for training. Since scaling the variables also impacts their estimated coefficients and average partial effects of all other features, appropriate re-scaling is necessary to evaluate the true effects in units of the real data. Other issues can arise from the incorrect weight initialization and wrong setting of hyperparameters such as learning rate or additional parameters introduced by more sophisticated variations of stochastic gradient descent, like Adam [23] or AdaGrad [12].

We estimate our model in TensorFlow by following three steps. At first, a model object needs to be created, where the exact model specification is chosen, e.g. 'logit' or 'probit'. At this stage, further customization is possible, see chapter 4.5 for details. The next step is to compile the model including all available keyword arguments of the original Keras functionality like optimizer selection or callback settings. Finally, model fitting is initialized by calling the fit method. Again all keyword arguments of the respective Keras functionality are preserved here. A code snippet exemplifying these steps is displayed in Figure 3. As an additional feature we also allow to specify the floating point repre-

```
logitmodel=Pooled_logit_probit(link='logit')
logitmodel.compile(input_dim=10 ,float64=True, optimizer='Adam')
logitmodel.fit(dataset, epochs=1, batch_size=128)
```

Figure 3: Code snippet for creating and fitting a pooled regression model.

sentation to be used when compiling the model. By default, Keras models are based on float32 in order to fit higher amounts of data into memory at the cost of lower numerical resolution compared to float64, which is the default representation in numpy. However, a higher numerical precision can be beneficial for the task of statistical inference on very large datasets. Many statistical quantities of interest, like goodness of fit characteristics or p-values, require to condense information of the entire dataset into a single mean-

ingful number. If the underlying dataset is very large, rounding errors resulting from a huge amount of individual floating point operation are accumulated which could cause imprecise final results. Such issues can easily be mitigated, by choosing a floating point representations with higher numerical precision. But since this also slows down training speed and doubles memory requirements, we allow the user to choose between float32 and float64 by himself based on his individual use case.

TensorFlow also started to support additional numerical representations relying on float16 which could speed up computations considerably [22]. But since this also further limits the numerical precision of all results, this is not recommended for conducting statistical inference at scale.

4.4 Statistical summary for Pooled Regression in TensorFlow

As one of our main achievements, we extend the existing Tensorflow capabilites by providing also a statistical summary of the estimated model to further evaluate the results and allowing causal inference. To get such a summary, the user can simply call the get_statistical_summary method of a fitted model as shown in Figure 4.

```
logitmodel.get_statistical_summary(robust=False, cofi_level=0.95)
```

Figure 4: Code snippet for getting the statistical summary.

The method further allows the user to specify the desired level of confidence for the computed confidence intervals. Furthermore, we offer the possibility to estimate robust sandwich errors, also known as Eicker-Huber-White standard errors [15][21][28], to correct for potential heteroskedasticity, as implemented in [32]. Running this method will compute the statistical summary and yield an output like in Figure 10:

```
Deviance: 937583.4917471232
Null Deviance: 1386294.3575198718
p-value of LR test: 0.0
McFadden-R2: 0.3236764712622129
McFadden-R2 adj.: 0.323669979134512
AIC: 937605.4917471232
BIC: 937735.4623632608
Significance codes: 0. < *** < 0.001 < ** < 0.01 < * < 0.05 < . < 0.1 <  < 1
```

	coefficent	standard error	z_score	p-value	Confidence Interval	Significance
(intercept)	-0.198416	0.002538	-78.182719	0.0	[-0.20339, -0.193442]	***
1	-0.036824	0.004025	-9.148559	0.0	[-0.044713, -0.028935]	***
2	-0.849229	0.004149	-204.695446	0.0	[-0.85736, -0.841097]	***
3	1.175609	0.004254	276.380048	0.0	[1.167272, 1.183946]	***
4	-0.921350	0.004172	-220.837964	0.0	[-0.929527, -0.913172]	***

Figure 5: Statistical summary for pooled regression in TensorFlow.

The summary includes statistical properties of the model such as different goodness of fit characteristics (deviance, null deviance, p-value of a likelihood ratio test, McFadden

pseudo R2's) and information criteria (Akaike and Bayesian). These quantities allow the user to asses how well a model describes the data and to compare different models in this regard. We also displays all coefficients with their respective standard errors and p-values, which are needed to conduct causal inference. All information are also saved as an attribute of the model object and thus can be easily accessed for potential further usage.

4.4.1 Mathematical details

From a mathematical perspective, all goodness of fit characteristics rely on the value of the log likelihood at the estimated coefficients and can be computed using their respective formula. Let $n \in \mathbb{N}$ be the number of observations, L_M be the likelihood of the fitted model with p parameters, L_S be the likelihood of the saturated model yielding perfect prediction and L_0 be the likelihood of the null model constantly predicting $\bar{y} = \frac{1}{n}\sum_{i=1}^{N} y_i$. Note that for the Bernoulli likelihood function used in case of binary response variables, $L_S = 1$ holds.

$$
\begin{aligned}
&Deviance = -2(\log L_M - \log L_S) = -2(\log L_M) \\
&Nulldeviance = -2(\log L_0 - \log L_S) = -2(\log L_0) \\
&\text{LR-test statistic: } LR = -2\log\left(\frac{L_M}{L_0}\right) = -2(\log L_M - \log L_0) \quad \text{where} \quad LR \sim \chi^2(p) \\
&R^2_{McFadden} = 1 - \frac{\log L_M}{\log L_0} \qquad R^2_{McFadden,adj} = 1 - \frac{\log L_M - p}{\log L_0} \\
&\text{Akaike Information Criteria:} \quad AIC = 2p - 2\log L_M \\
&\text{Bayesian Information Criteria:} \quad BIC = p\log n - 2\log L_M
\end{aligned}
$$

In order to get standard errors of the model coefficients $\beta \in \mathbb{R}^p$ we need to compute the covariance matrix at the maximum likelihood estimate $\hat{\beta}_{ML}$, which is the inverse of the Fisher information matrix $\mathcal{I}(\hat{\beta}_{ML})$. Let $\ell(\beta)$ be the log likelihood function depending on β and $\frac{\partial}{\partial\beta}\ell(\beta)$ be its gradient. Due to the concavity of the Bernoulli log likelihood and the existence of an optimum, we know that at $\hat{\beta}_{ML}$ it holds $\frac{\partial}{\partial\beta}\ell(\hat{\beta}_{ML}) = 0$. Thus:

$$
\mathcal{I}(\hat{\beta}_{ML}) = Var\left[\frac{\partial}{\partial\beta}\ell(\hat{\beta}_{ML})\right] = E\left[\frac{\partial}{\partial\beta}\ell(\hat{\beta}_{ML})\frac{\partial}{\partial\beta}\ell(\hat{\beta}_{ML})^T\right] = -E\left[\frac{\partial^2}{\partial\beta\partial\beta}\ell(\hat{\beta}_{ML})\right]
$$

Here, the last equality follows under certain regularity conditions which are satisfied in our model specifications, see [7] section 5.6.3 for details. Therefore, the Fisher information equals the negative expected Hessian matrix. With true coefficient vector β^* it holds asymptotically, that $\hat{\beta}_{ML} \sim \mathcal{N}(\beta^*, \mathcal{I}(\beta^*)^{-1})$ and standard errors can be computed by taking the square root of the diagonal entries of the covariance matrix $\mathcal{I}(\hat{\beta}_{ML})^{-1}$.

In case of logistic regression on observations $\{x_i \in \mathbb{R}^p\}_{i=1}^n$ we get the following expression for the fisher information:

$$
\mathcal{I}_{logit}(\beta) = \sum_{i=1}^{n} \frac{\exp(\beta^T x_i)}{(1+\exp(\beta^T x_i))^2} x_i x_i^T
$$

For the respective probit regression we get with standard normal cdf Φ and density ϕ:

$$\mathcal{I}_{probit}(\beta) = \sum_{i=1}^{n} \frac{\phi(\beta^T x_i)^2}{\Phi(\beta^T x_i)(1 - \Phi(\beta^T x_i))} x_i x_i^T$$

4.4.2 Computational details

The computation of a statistical model summary like presented above requires to read and process the entire dataset. In order to do this at scale, ensuring that it works for extra-large out-of-memory datasets, we compute all necessary quantities in a batchwise fashion and reduce all partial results in the end to receive the final result. With TensorFlow, we can accelerate these partial computations by executing them on a GPU or even distribute them across multiple GPUs if available.

Since reading batches of data from disk is rather slow, we utilize the TensorFlow Dataset API to already prefetch the next batch to main memory while GPUs are processing the current batch. All quantities of interest rely on the log likelihood itself or its Hessian matrix evaluated at the maximum likelihood estimate. Since both expressions are algebraically already formulated as sum over each individual element of the dataset, batching and distributing is straightforward. The calculation of the Hessian matrix $H(\beta)$ of the estimated coefficients β is visualized in Figure 6, where $\ell_i(\beta)$ describes the value of the log likelihood function of data point i over batches B_1, B_2... Note that all partial results are stored in main memory to account for in general more restrictive GPU memory constrains and reduced to the final result on the CPU.

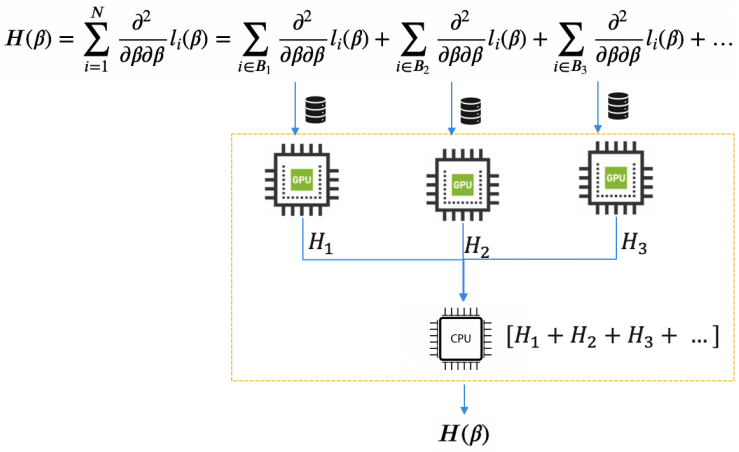

Figure 6: Distributed Hessian computation on multiple GPUs

Since we utilize CPUs and GPUs simultaneously to perform computations, it is worth mentioning that the results retrieved from both kinds of devices cannot be expected to be exactly identical in general [29]. The root cause for this can also be attributed to rounding errors in floating point arithmetic and thus is also alleviated by using float64.

4.5 Beyond logit and probit using automatic differentiation

Computing standard errors of parameters requires to evaluate the Hessian matrix of log likelihood function at the maximum likelihood estimate. For this reason and in line with traditional statistical software, we have to individually derive the exact analytic expressions of Hessian matrix for a logit and a probit model to be used in the code.

The main drawback of this approach is that each new model specification, like a link function other than logit or probit, would need additional mathematical considerations as well as time consuming extra coding effort. For example, customizing a link function with the glm package in R requires the user to specify and implement not only the link itself, but also three supplementary functions[1]. This is quite cumbersome, considering that the logic of the computation is exactly the same for different links.

We develop simple way to generalize our scalable code basis for an arbitrary link functions by calculating the specific Hessian matrix automatically without extra code using TensorFlow's automatic differentiation feature. Automatic Differentiation [4] is an algorithmic technique that enables the calculation of exact partial derivatives in a numerically stable and efficient way and is therefore the state of the art procedure to perform this task in modern machine learning [5]. Before execution, TensorFlow translates the underlying code into a computational graph where each node represents a basic mathematical operation. For each of these basic operations available, TensorFlow also knows the exact expression of the partial derivatives with respect to different input values. Thus, overall derivatives can be simply computed with a forward pass through the graph storing all partial results at the nodes followed by a backwards pass applying the chain rule of differentiation. This procedure is exemplified in Figure 7.

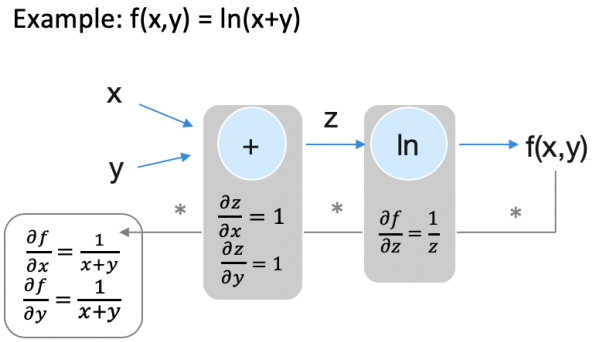

Figure 7: Example of automatic differentiation

Leveraging automatic differentiation enables us to fit and compute the summary of models based on arbitrary link functions with very few extra lines of code offering maximum flexibility and complete scalability to the user. Figure 8 shows how to create a scalable binary regression model using the *cloglog* link function, which needs to be first implemented and then referred to in the model constructor. In fact, this idea can be further used to also fit and evaluate entirely different model specifications like other variants of

[1] for details see: https://stat.ethz.ch/R-manual/R-devel/library/stats/html/make.link.html

generalized linear models by additionally specifying the respective negative log likelihood function. As an example, the code needed for simple Poisson regression is depicted in Figure 9.

```
def cloglog_activation(x):
    y= 1-tf.math.exp(-tf.math.exp(x))
    y = tf.clip_by_value(y, 1e-7, 1.-1e-7)
    return y

cloglogmodel=Pooled_logit_probit(link=cloglog_activation)
cloglogmodel.compile(input_dim=37 ,float64=True, optimizer='RMSProp')
cloglogmodel.fit(dataset, prepro_batch_size=100000, epochs=8, training_batch_size=1024)
```

Figure 8: Code for implementing a cloglog model

```
def poisson_link(x):
    return tf.math.exp(x)

def poisson_likelihood(labels, predicted):
    return -(labels*tf.math.log(predicted +1e-7) - predicted)

poissonmodel=Pooled_logit_probit(link=poisson_link, loss=poisson_likelihood)
poissonmodel.compile(input_dim=10 ,float64=True, optimizer='SGD')
poissonmodel.fit(poisson_dataset, prepro_batch_size=100000, epochs=10, training_batch_size=32)
```

Figure 9: Code for implementing a poisson model

Offering such amount of flexibility however, comes with costs. At first, computing the Hessian matrix automatically instead of using the analytic expression will take usually more time. Secondly, although automatic differentiation is in general numerically stable, additional instabilities could arise from the mathematical form of the used link or log likelihood functions. To mitigate such issues, feature scaling, clipping specific values and adding small epsilon values at critical computation steps can be applied to remain in a numerically stable range. In case of the *cloglog* link for binary regression model with cross entropy log likelihood structure, such instabilities could occur for predicted probabilities that are too close to one or zero. To avoid this, scaling the features to be in the range of -1 and 1 as well as clipping the output of the link function performs very effectively in our simulations. For the Poisson log likelihood, we add a small term inside the logs to avoid the singularity at zero.

Note that in this setting, standard errors are computed using the observed information $\mathcal{J}(\beta) = -\frac{\partial^2}{\partial\beta\partial\beta}\ell(\beta)$ rather than using the Fisher information matrix $\mathcal{I}(\beta) = E\left[\mathcal{J}(\beta)\right]$. Both methods are used in different available software packages and evidence suggests, that the observed information might even be a better estimate of the asymptotic covariance [14].

5 Fixed Effects Models

Since we are dealing with panel data, we have multiple observations for each entity. Instead of just pooling data we can use this data structure to infer the unobserved effect of each observed entity and control for it. Here "fixed effects" (unit specific intercepts) is used to account for unmodeled heterogeneity in grouped data analyses. In this section,

we present the fixed effect logit model for dealing with unobserved effects in panel data. This is a modification of the standard logit model where a constant unobserved effect is added to each group of observations. It is a well known model in econometrics that can be used to model a binary response in cases where we observe the same entity (person, corporation, country...) at multiple times and we want to control for characteristics of this entity that are unobserved or when the entities we are observing can be divided into groups and we want to control for effects of being in this group on our response. We begin by introducing the mathematical setup of the model.

5.1 Model setup

We start by introducing the notation we use to describe our dataset. The number of groups in our dataset are labeled as $I \in \mathbb{N}$ and the number of observations in each group as $T \in \mathbb{N}^I$. $\overline{T}$ will denote the average group size. $\frac{1}{I}\sum_{i=1}^{I} T_i$. Each observation contains $R \in \mathbb{N}$ covariate features. Our observed features are given as $x = (x_1, x_2, ..x_I)$ where $\forall i \leq I \ x_i \in \mathbb{R}^{T_i \times R}$ contains of observations for one group. We also have a response dataset $y = (y_1, y_2, ..y_I)$ where $\forall i \leq I \ y_i \in \mathbb{R}^{T_i}$ and $\forall t \leq T_i \ y_{it}$ is the observed response corresponding to the feature vector x_{it}.

We treat y as a draw from an I-dimensional vector $Y = (Y_1, Y_2, \dots Y_I)$ where each Y_i is a T_i-dimensional random vector. We are assuming that the distribution of each response variable Y_{it} can be modelled as a Bernoulli distribution where the probability of success is given as a sum of a linear combination of features and the group fixed effects plugged into the logistic function. Assuming that $\beta \in \mathbb{R}^R$ is the vector of coefficients in the linear combination and $\alpha \in \mathbb{R}^I$ is the vector of fixed effects for each group, we say that $\mathbb{P}(Y_{it} = 1 \mid \beta, \alpha_i, x_{it}) = \sigma(\alpha_i + x_{it}\beta)$, where σ is the logistic (sigmoid) function given by $\sigma(x) = \frac{e^x}{1+e^x}$. Note that the only difference between this model and well-known logistic model is the inclusion of the fixed effect α_i term.

To do any reasonable inference, we also assume that $(Y_{it_1} \mid \beta)$ is independent of $(Y_{jt_2} \mid \beta)$ if $i \neq j$ and that $(Y_{it_1} \mid \beta, \alpha_i)$ is independent of $(Y_{it_2} \mid \beta, \alpha_i)$ if $t_1 \neq t_2$. These assumptions are in general hard to verify on real data. However the fact that our observations happen at different times and are operationally not expected to have much influence on each other, we believe they are met in our dataset to a reasonable degree. We also make the assumption that the observed values our features are all linearly independent to guarantee uniqueness of MLE estimators (to be introduced in the following sections). This is easy to verify by putting all x_{it} vectors into a matrix and verifying that the rank of this matrix is equal to R. A final condition to guarantee uniqueness of MLE is that for every feature there is at least one group in which the feature has at least two different values. If a feature is invariant in every group there is no unique solution. This condition is also easy to verify, and can be simply met by removing invariant features and implicitly including them in the fixed effects.

Our goal is to construct consistent estimators $\hat{\beta}$ and $\hat{\alpha}$ for β and α. If our assumptions for the model hold these estimators give us an understanding as to how each variable affects the probability of getting a positive response. We use two different MLE estimators to

find reasonable $\hat{\beta}$ and $\hat{\alpha}$ as described in the following sections.

5.2 Unconditional likelihood method

The most intuitive way to find our estimators is to find values of $\hat{\beta}$ and $\hat{\alpha}$ that maximize the likelihood of our model. As we are working with the pure likelihood with no additional conditioning, we will call this the "Unconditional likelihood" (to contrast to the Conditional likelihood in the next section). Implementation-wise this means treating group characteristics as separate variables and dummy-encoding them into our features. Compared to the alternative, the Conditional likelihood method, this method requries less computation when dealing with large group sizes ($\mathcal{O}(T)$ vs $\mathcal{O}(T^2)$), which is crucial for our goal of scaling our estimator to work on giant datasets. However, it suffers from bias due to incidental parameters. The details of why it is biased and which methods exist to deal with the bias are discussed in the next section. We can easily derive the likelihood function for the fixed effect model using the independence assumptions:

$$L(\beta, \alpha, y) = \mathbb{P}(Y = y \mid \beta, \alpha) = \prod_{i=1}^{I} \prod_{t=1}^{T_i} \mathbb{P}(Y_{it} = z_t \mid \beta, \alpha_i) =$$

$$\prod_{i=1}^{I} \prod_{t=1}^{T_i} \sigma(\alpha_i + x_{it}\beta^T)^{y_{it}}(1 - \sigma(\alpha_i + x_{it}\beta^T))^{1-y_{it}}$$

Then the log likelihood is given by

$$\ell(\beta, \alpha, y) = \sum_{i=1}^{I} \sum_{t=1}^{T_i} ((y_{it}(\alpha_i + x_{it}\beta^T)) - \ln(1 + exp(\alpha_i + x_{it}\beta^T))).$$

We can now define $(\hat{\beta}_{MLE}, \hat{\alpha}_{MLE}) = \underset{(\beta,\alpha)}{\operatorname{argmax}}(\ell(\beta, \alpha, y))$

It is proven in [8] that the likelihood function is strictly concave with respect to β and α under the linear independence and time-invairance assumptions. Hence, a solution for this equation exists and is unique.

5.2.1 Computing the estimator

As the log likelihood function is strictly concave, there is only one local extremum and it is at the global maximum, so $(\hat{\beta}_{MLE}, \hat{\alpha}_{MLE})$ satisfies $\dot{\ell}(\hat{\beta}_{MLE}, \hat{\alpha}_{MLE}, y) = 0$. As the likelihood equation is concave and smooth the Newton-Rapshod method converges quickly. Thus if we define $(\beta^{(0)}, \alpha^{(0)}) = 0$ and $(\beta^{(n+1)}, \alpha^{(n+1)}) = (\beta^{(n)}, \alpha^{(n)}) - H^{-1}\dot{\ell}(\beta^{(n)}, \alpha^{(n)}, y)$ where H is the Hessian of the log likelihood ℓ at the point $(\beta^{(n)}, \alpha^{(n)}, y)$ we get a sequence that converges to $(\hat{\beta}_{MLE}, \hat{\alpha}_{MLE})$.

Computing this requires inverting the $(R + I) \times (R + I)$ Hessian. In general matrix inversion is a computationally costly process (as the Hessian is a Hermitian matrix it can be done by Cholesky decomposition in $\mathcal{O}((R + I)^3)$ time). As I, the number of groups, can get very large on large dataset (specifically we expect I to be $\mathcal{O}(N)$ on our dataset) this can present challenges. Luckily, if we decompose our Hessian,

$$H = \begin{bmatrix} H_{\beta,\beta} & H_{\beta,\alpha} \\ H_{\alpha,\beta} & H_{\alpha,\alpha} \end{bmatrix},$$

where $H_{\beta,\beta} = (\frac{\partial^2 \ell}{\partial \beta_i \beta_j})_{i \leq R, j \leq R}$, $H_{\alpha,\beta} = (\frac{\partial^2 \ell}{\partial \alpha_i \beta_j})_{i \leq I, j \leq R}$, $H_{\beta,\alpha} = H_{\alpha,\beta}^T$, $H_{\alpha,\alpha} = (\frac{\partial^2 \ell}{\partial \alpha_i \alpha_j})_{i \leq I, j \leq I}$
we notice that if $i \neq j$, $\frac{\partial^2 \ell}{\partial \alpha_i, \alpha_j} = \frac{\partial}{\partial \alpha_i}(\sum_{t=1}^{T_j}(y_{jt} - \frac{exp(\alpha_j + x_{jt}\beta^T)}{1 + exp(\alpha_j + x_{jt}\beta^T)})) = 0$.
So $H_{\alpha,\alpha}$ is diagonal. Taking advantage of this, and utilizing Schur's complement to write out the inverse we can get an exact formula for updating (β, α) that requires only the inversion of a $R \times R$ matrix and the I-fold addition of R-dimensional vectors leading to a complexity of $\mathcal{O}(IR)$. The details of this approach and the resulting update formulas can be found in Appendix A1 of [27].This appendix also contains closed form expressions for the gradient and the Hessian. Using these update formulas we have a computationally efficient method for finding $(\hat{\beta}_{MLE}, \hat{\alpha}_{MLE})$.

Numerically speaking, inverting a matrix is not in general a stable operation. If a matrix is badly conditioned, inverting it can explode numerical errors. As we use matrix inversion as an integral component of both methods this is a significant problem. Fortunately, the matrix we are inverting is a Hessian of a concave function, and thus positive definite. This means that instead of regular matrix inversion we can use a LDL decomposition to find the inverse. This greatly improves numerical stability.
If A is a positive definite matrix then a decompostion $A = LDL^T$ (where L is a lower unit triangular matrix and D is a positive diagonal matrix) exists, is unique and we call it a LDL decompostion. Details of how to implement and proofs of numerical stability when solving a system with a LDL decomposed matrix can be found in [20].
However, due to floating-point rounding errors the computed version of the Hessian might not in reality be positive definite. To be able to still run a numerically stable algorithm in this scenario we implement the GMW81 Modified Cholesky Algorithm described in [26]

5.2.2 Incidental parameters problem

Our model falls into the nuisance parameter setup from [24], treating α as a nuisance parameter in estimating β we can conclude that this estimator is biased and inconsistent when N $\to \infty$ and T remains bounded. Specifically it can be shown that $\mathbb{E}[\hat{\beta}_{MLE}] = \beta_0 + \frac{B}{T} + \sigma(\frac{1}{(T)^2})$ for some constant B. The reason behind it is that when the number of intercepts to be estimated goes to infinity as the number of observations goes to infinity (Eg. behavioral panel case), we get the "incidental parameters" problem; in this situation standard maximum likelihood results do not hold and maximum likelihood estimators may not be consistent. In fact, the variance of $\hat{\alpha}_{MLEi}$ is $\mathcal{O}(\frac{1}{T_i})$. As it is impossible to separate α_i and β from the non-linear likelihood function any variance $\hat{\alpha_{MLEi}}$ results variance in the estimate for $\hat{\beta_{MLE}}$ from that observation. There are two known approaches to correcting bias both described in [18]. Jackknifing and an analytical method.

Jackknife Jackknifing is based on the idea that the size of the dominant term in the bias expression $(\frac{B}{T_i})$ is inverse proportional to group size. Therefore. halving the number of observations in each group would result in doubling the term. Thus, if for each dataset group (x_i, y_i) we randomly sample half the group and add it to a new dataset (x_i^1, y_i^1) and then add the other half to (x_i^2, y_i^2) we can get $\hat{\beta}_{MLE}^1$ and $\hat{\beta}_{MLE}^2$ as Unconditional MLE estimates of β_0 on half-sets (x^1, y^1) and (x^2, y^2). Average group size for each of these half-sets satisfies: $\overline{T_1} = \overline{T_2} = \frac{\overline{T}}{2}$ and then it follows that:

$\mathbb{E}[\hat{\beta}^1_{MLE}] = \mathbb{E}[\hat{\beta}^2_{MLE}] = \beta_0 + \frac{2B}{\overline{T}} + o(\frac{1}{\overline{T}^2})$

so if $\hat{\beta}^{JN}_{MLE} = 2\hat{\beta}_{MLE} - \frac{1}{2}(\hat{\beta}^1_{MLE} + \hat{\beta}^2_{MLE})$

then $\mathbb{E}[\hat{\beta}^{JN}_{MLE}] = 2\mathbb{E}[\hat{\beta}_{MLE}] - \frac{1}{2}(\mathbb{E}[\hat{\beta}^1_{MLE}] + \mathbb{E}[\hat{\beta}^2_{MLE}]) = \beta_0 + o(\frac{1}{\overline{T}^2})$

This does not remove the bias completely, but it reduces the effect of the bias by an order of magnitude. It can be seen in [18] that this correction performs well on a Monte-Carlo experiment even with a strongly bounded group size (≤ 8).

Based on the idea of halving the panel data, we opt for a cross-over jackknife scheme. As our estimator is sensitive to sampling methods we attempt to avoid instability by attempting to capture the data structure in both subsets. Specifically, we divide each panel in our dataset into the following two subpanels.

$$S_1 = \{(i,t) : i \leq \lceil I/2 \rceil, t \leq \lceil T_i/2 \rceil\} \cup \{(i,t) : i \leq \lfloor I/2 + 1 \rfloor, t \leq \lfloor T_i/2 + 1 \rfloor\}$$

$$S_2 = \{(i,t) : i \leq \lceil I/2 \rceil, t \geq \lceil T_i/2 \rceil\} \cup \{(i,t) : i \geq \lfloor I/2 + 1 \rfloor, t \leq \lfloor T_i/2 + 1 \rfloor\}$$

,where $\lceil . \rceil$ and $\lfloor . \rfloor$ denote the ceiling and floor function. Forming an estimator $\beta_{jack} := 2\hat{\beta} - (\beta_{S_1} + \beta_{S_2})/2$ removes bias both bias terms in large samples.

Asymptotic Analysis The analytical method consists of deriving an estimator $\hat{B}$ for the B in the bias term using a modification of the MLE and Bartlett identities and then subtracting it:

$\hat{\beta}^{AN}_{MLE} = \hat{\beta}_{MLE} - \frac{\hat{B}}{\overline{T}}.$, assuming $\mathbb{E}[\hat{B}] = B$, we get:

$\mathbb{E}[\hat{\beta}^{AN}_{MLE}] = \mathbb{E}[\hat{\beta}_{MLE}] - \frac{\mathbb{E}[\hat{B}]}{\overline{T}} = \beta_0 + o(\frac{1}{\overline{T}^2})$

The formula for the estimator and the process of deriving it are given in [18] chapter 4. It preforms similarly to the Jackknife estimator in the aforementioned Monte-Carlo experiments. In particular, under some regularity conditions, [p.11 [18]] concluded that analytical bias correction works asymptotically as well.

Given that the Analytical method is to our knowledge so far only developed for use in balanced panels, and the Jackknifing method works for unbalanced panels (panels that are not all of the same size) as well, and that Jackknifing works no matter which link function is used we opt to use the Jackknifing method in our implementation.

5.3 Conditional likelihood method

Another way to estimate parameters in Fixed effects model is to use conditioning to get a likelihood equation without an α term. Conceptually it similar to demeaning in the case of fixed effects linear regression models[27]. In a logistic model this is done by conditioning Y_i on $\sum_{t=1}^{T_i} Y_{it}$ and trying to maximize it's likelihood. More formally:
Take any $i \leq I$ and any $z \in \{0,1\}^{T_i}$, then by independence:

$$\mathbb{P}(Y_i = z \mid \beta, \alpha_i) = \prod_{t=1}^{T_i} \mathbb{P}(Y_{it} = z_t \mid \beta, \alpha_i) = \frac{exp(\alpha_i \sum_{t=1}^{T_i} z_t + \sum_{t=1}^{T_i} z_t x_{it}\beta^T)}{\prod_{t=1}^{T_i}(1 + exp(\alpha_i + x_{it}\beta^T))}$$

Now we define $k = \sum_{t=1}^{T_i} z_t$ and $Z^k = \{z^k \in \{0,1\}^{T_i} \mid \sum_{t=1}^{T_i} z_t^k = k\}$, we can introduce an ordering on this set to get $z^k_{(1)}, z^k_{(2)}, \dots z^k_{\binom{k}{T_i}}$.

$$\text{Then: } \mathbb{P}(\sum_{t=1}^{T_i} Y_{it} = k \mid \beta, \alpha_i) = \sum_{j=1}^{\binom{k}{T_i}} \mathbb{P}(Y_i = z^k_{(j)} \mid \beta, \alpha_i) = \frac{\sum_{j=1}^{\binom{k}{T_i}} exp(\alpha_i \sum_{t=1}^{T_i} z^k_{(j)t} + \sum_{t=1}^{T_i} z^k_{(j)t} x_{it}\beta^T)}{\prod_{t=1}^{T_i}(1 + exp(\alpha_i + x_{it}\beta^T))}$$

$$\text{Then : } \mathbb{P}(Y_i = z \mid \sum_{t=1}^{T_i} Y_{it} = k, \beta, \alpha_i) = \frac{\mathbb{P}(Y_i = z \mid \beta, \alpha_i)}{\mathbb{P}(\sum_{t=1}^{T_i} Y_{it} = k \mid \beta, \alpha_i)} = \frac{exp(\sum_{t=1}^{T_i} z_t x_{it}\beta^T)}{\sum_{j=1}^{\binom{k}{T_i}} exp(\sum_{t=1}^{T_i} z^k_{(j)t} x_{it}\beta^T)}$$

Note that this expression does not depend on α_i ($\sum_{t=1}^{T_i} Y_{it}$ is sufficient for it) so we can define a conditional likelihood function

$L_C(\beta, y) = \prod_{i=1}^{I} \mathbb{P}(Y_i = y_i \mid \sum_{t=1}^{T_i} Y_{it} = \sum_{t=1}^{T_i} y_{it}, \beta)$ and the log likelihood:

$\ell_C(\beta, y) = \sum_{i=1}^{I} \ln \mathbb{P}(Y_i = y_i \mid \sum_{t=1}^{T_i} Y_{it} = \sum_{t=1}^{T_i} y_{it}, \beta)$ and $\hat{\beta}_{CMLE} = \underset{\beta}{\text{argmax}} \ell_C(\beta, y)$

It is proven in [8] this likelihood function is also concave, so it gives a unique solution, but also that this method gives a unbiased and consistent estimator for β_0, unlike the unconditioned MLE method.

5.3.1 Computing the estimator

Just like in the unconditional case, the log likelihood function is concave, and there is only one local extremum which is at the global maximum, so $\hat{\beta}_{CMLE}$ satisfies $\dot{\ell_C}(\hat{\beta}_{CMLE}, y) = 0$. Again the likelihood equation is concave and smooth the Newton-Rapshod method converges quickly.

So analogously we can define $\beta^{(0)} = 0$ and the recursion: $\beta^{(n+1)} = \beta^{(n)} - H_C^{-1}\dot{\ell}_C(\beta^{(n)}, y)$. where H_C is the Hessian of the conditional likelihood at $(\beta^{(n)}, y)$. for computing a sequence that converges to $\hat{\beta}_{CMLE}$.

If we define:

$$\ell^i_C(\beta, y) = \ln \mathbb{P}(Y_i = y_i \mid \sum_{t=1}^{T_i} Y_{it} = \sum_{t=1}^{T_i} y_{it}, \beta, \alpha_i) \qquad t_i = \sum_{j=1}^{T_i} y_{it}$$

$$d0^i = \sum_{j=1}^{\binom{t_i}{T_i}} exp(\sum_{t=1}^{T_i} z_{(j)t}^{t_i} x_{it} \beta^T) \qquad d1_l^i = \frac{\partial d0^i}{\partial \beta_l} = \sum_{j=1}^{\binom{t_i}{T_i}} (\sum_{t=1}^{T_i} z_{(j)tl}^{t_i} x_{itl}) exp(\sum_{t=1}^{T_i} z_{(j)t}^{t_i} x_{it} \beta^T)$$

$$d2_{lk}^i = \frac{\partial d1_l^i}{\partial \beta_k} = \sum_{j=1}^{\binom{t_i}{T_i}} (\sum_{t=1}^{T_i} z_{(j)tl}^{t_i} x_{itl})(\sum_{t=1}^{T_i} z_{(j)tk}^{t_i} x_{itk})(exp \sum_{t=1}^{T_i} z_{(j)t}^{t_i} x_{it} \beta^T)$$

Then simple differentiation yields: $\ell_C^i(\beta, y) = \sum_{t=1}^{T_i} y_{it} x_{it} \beta^T - \ln d0^i$

$$\frac{\partial \ell_C^i}{\beta_l}(\beta, y) = \sum_{t=1}^{T_i} y_{itl} x_{itl} - \frac{d1_l^i}{d0^i} \qquad \frac{\partial^2 \ell_C^i}{\beta_l \beta_k}(\beta, y) = \frac{d2_{lk}^i}{d0^i} - \frac{d1_l^i d1_k^i}{(d0^i)^2}$$

This is worrying as the expression for our likelihood, gradient and Hessian contain sums of $\binom{t_i}{T_i}$ terms. Even for limited group sizes this quickly becomes unfeasible to compute brute-force (eg. $t_i = 10, T_i = 100 \implies \binom{t_i}{T_i} \approx 10^{18}$)
Since many of these terms share common factors, a dynamic recursion can be defined that can find $d0^i$, $d1^i$ and $d2^i$ terms in $\mathcal{O}(t_i T_i R^2)$ time. Recursively define:

$$d0^i(0, n) = 1 \qquad d0^i(d, n) = d0^i(d-1, n-1) exp(x_{in} \beta^T) + \mathbb{1}_{d<n} d0^i(d, n-1)$$

differentiating both sides by β_l yields:

$$d1_l^i(d, n) = d0(d-1, n-1) x_{inl} exp(x_{in} \beta^T) + \mathbb{1}_{d<n} d1_l^i(d, n-1) + \mathbb{1}_{d>1} d1_l^i(d-1, n-1) exp(x_{in} \beta^T)$$

differentiating both sides by β_k yields:

$$d2_{lk}^i(d, n) = d0(d-1, n-1) x_{inl} x_{ink} exp(x_{in} \beta^T) + \mathbb{1}_{d<n} d2_{lk}^i(d, n-1)$$
$$+ \mathbb{1}_{d>1} exp(\beta_n x_{in})(d2_{lk}^i(d-1, n-1) + d1_l^i(d-1, n-1) x_{ink} + d1_k^i(d-1, n-1) x_{inl})$$

It can be easily checked that $d0^i = d0^i(t_i, T_i)$, $d1_l^i = d1_l^i(t_i, T_i)$ and $d2_{lk}^i = d2_{lk}^i(t_i, T_i)$
As each recursion has $\frac{t_i T_i}{2}$ states we see that this is the number of steps required to calculate any single value. As there are R^2 components of $d2^i$ we need $\mathcal{O}(t_i T_i R^2)$ per group to calculate the Hessian. As $t_i \leq T_i$ we get a worst case complexity of $\mathcal{O}(\|T\|_2^2 R^2)$. For a bounded vector T this can be computationally efficient. But for $\|T\|_\infty \to \infty$ we will need to rely on approximations of the log likelihood, gradient and Hessian.

5.3.2 Approximating the log likelihood

Breslow (see [6]) and Peto (see [16]) suggested an approximation computing $d0^i$ as $\left(\sum_{t=1}^{T_i} exp(x_{it} \beta^T) \right)^{t_i}$. This approximation will break down when t_i are large relative to the size of the observations of the group, T_i, and then tends to yield estimates of β, which are biased toward 0.

Efron(see [13]) suggests an even closer approximation to the likelihood, computing $d0^i$ as $\Pi_{r=1}^{t_i} \Pi_{r=1}^{t_i} \left(\sum_{l=1}^{t_i} (1 - \frac{r-1}{t_i}) exp(x_{il} \beta^T) \right)$

Like the Breslow approximation, Efron's[13] method yields estimates of β, which are biased toward 0 when there are many instances within a group. However, the Efron approximation is much faster than the exact methods and tends to yield much closer

estimates than the Breslow approach(see [13]).

Hertz [19], points out using simulated data without censoring in Cox model that Breslow approximation tends to underestimate the true β. As the ties become heavier, the bias of these approximations increases. The Efron [13] approximation performs far better than the other two, particularly with moderate or heavy ties; even with n = 25 in each group, the bias is under 2 percent, and for sample sizes larger than 50 per group, it is less than 1 percent. Although the Breslow approximation is the default in many standard software packages, the Efron method for handling ties is to be preferred, particularly when the sample size is small.

In our implementation, correctness is a more important factor than computational complexity(as this could be mitigated with a larger cluster of computers), so we opt to use the exact method.

5.4 Average partial effects

The formula for the partial effect with respect to an observation x_{it} in the Fixed effects model is given by:

$$\frac{\partial \mathbb{P}(Y_{it} = 1 \mid \beta_0, \alpha_{0i}, x_{it})}{\partial x_{it}} = f(\alpha_i + x_{it}\beta)\beta$$

Where f is is the logistic pdf.

So the average partial effects are given as an average of these effects

$$APE = \frac{1}{N\overline{T}} \sum_{i=1}^{I} \sum_{t=1}^{T_i} f(\alpha_i + x_{it}\beta)\beta$$

We can get an estimator $\hat{APE}$ by plugging in estimators $\hat{\beta}$ and $\hat{\alpha}$

$$\hat{APE} = \frac{1}{N\overline{T}} \sum_{i=1}^{I} \sum_{t=1}^{T_i} f(\hat{\alpha}_i + x_{it}\hat{\beta})\hat{\beta}$$

The consistency of this estimator depends on the consistency of $\hat{\beta}$ and $\hat{\alpha}$. Maximizing the unconditional likelihood results in a biased $\hat{\beta}$ and an inconsistent $\hat{\alpha}$ while maximizing the conditional one leaves us with no $\hat{\alpha}$ estimator. If we knew the data-generating β we could estimate each α_i as and MLE estimator of the unconditional likelihood of group i with β fixed.
$\hat{\alpha_{i\beta}} = \underset{\alpha_i}{\operatorname{argmax}}(\ell_i(\alpha_i, \beta, y))$
This estimator is consistent as $T_i \to \infty$ if β is exact. This consistency property holds if we plug in a consistent estimator $\hat{\beta}$. The conditional method already produces one and the unconditional method does as well after application above described bias-correction methods. Therefore the following 3-step method:

1. Estimate a consistent $\hat{\beta}$ by conditional or bias-corrected unconditional MLE estimators

2. Plug in $\hat{\beta}$ to estimate $\hat{\alpha_\beta}$
3. Plug in $\hat{\beta}$ and $\hat{\alpha_\beta}$ to get $\hat{APE}$

results in a consistent estimator when $N \to \infty$ and for all i $T_i \to \infty$

5.5 Distributed computation to model Fixed Effects Logit

Looking at our formulas for the Hessian and gradient in the conditional model and in the unconditional model from [27] we see that they can be written as $H = \sum_{i=1}^{I} H_i$ and $g = \sum_{i=1}^{I} g_i$ where $\forall i$ the terms H_i and g_i can be computed only using data from group i. It is trivial to see that a similar approach can be used to compute APE's.
Conditional case:

$$H = \sum_{i=1}^{I} (\frac{d2_i}{d0_i} - \frac{d1_i}{d0_i}(\frac{d1_i}{d0_i})^T)$$

and

$$g = \sum_{i=1}^{I} (x_i y_i - \frac{d1_i}{d0_i})$$

Unconditional case(using notation from [27]):

$$H = H_{\beta\beta} - \sum_{i=1}^{I} (h^{-1}_{\alpha_i\alpha_i} h_{\beta\alpha_i} h^T_{\beta\alpha_i}) = \sum_{i=1}^{I}\sum_{t=1}^{T_i} x^T_{it} x_{it} p_{it}(1-p_{it}) - \sum_{i=1}^{I} (h^{-1}_{\alpha_i\alpha_i} h_{\beta\alpha_i} h^T_{\beta\alpha_i}) =$$
$$= \sum_{i=1}^{I} (\sum_{t=1}^{T_i} x^T_{it} x_{it} p_{it}(1-p_{it}) - h^{-1}_{\alpha_i\alpha_i} h_{\beta\alpha_i} h^T_{\beta\alpha_i})$$
$$g = g_\beta - \sum_{i=1}^{I} (h^{-1}_{\alpha_i\alpha_i} g_{\alpha_i} h_{\beta\alpha_i}) = \sum_{i=1}^{I}\sum_{t=1}^{T_i} x_{it} p_{it}(1-p_{it}) - \sum_{i=1}^{I} (h^{-1}_{\alpha_i\alpha_i} g_{\alpha_i} h_{\beta\alpha_i}) =$$
$$= \sum_{i=1}^{I} (\sum_{t=1}^{T_i} x_{it} p_{it}(1-p_{it}) - h^{-1}_{\alpha_i\alpha_i} g_{\alpha_i} h_{\beta\alpha_i})$$

This means that we can implement computing of Hessian and gradient contributions from each group independently and then simply add them all together. In terms of distributing computation this means that we can store most of our data on local machines and do computation for their contribution locally and then just broadcast the results and add them together. This process can easily be implemented as map-reduce procedure which makes it easy to use Spark packages for a simple and efficient distribution of computation allowing for clean scalabilty.

6 Results and Conclusion

As a result of our implementations in TensorFlow and Apache Spark, we are now able to compare the results of pooled and fixed effect regession on very large datasets. Figure 10 shows the retrieved APEs of pooled logistic, pooled probit and logistic regression with fixed effects. Note that for fixed effect models, all APEs of invariant features within groups are zero.

Coefficient	APE_logit	PE_at_mean_logit	APE_probit	PE_at_mean_probit	APE_uncond
0	-5.88E-10	-5.54E-10	-5.32E-10	-5.29E-10	0.00E+00
1	5.39E-04	5.08E-04	5.78E-04	5.75E-04	0.00E+00
2	-1.42E-04	-1.34E-04	-1.47E-04	-1.47E-04	-1.45E-04
3	7.69E-04	7.25E-04	8.07E-04	8.03E-04	2.77E-04
4	1.49E-06	1.40E-06	1.54E-06	1.53E-06	0.00E+00
5	2.57E-02	2.57E-02	2.46E-02	2.46E-02	0.00E+00

Figure 10: Average Partial Effects.

In this project, we introduce groundbreaking solutions for causality inference with big data technologies. From now on, we enable both the scientific community and business leaders to perform causality analysis on extra-large datasets with big data tools. Our unique package in TensorFlow allows for fitting pooled regressions with arbitrary link functions, utilizing parallel calculations on GPUs and automatic differentiation. The highly-scalable Apache Spark solution is the first existing approach for distributed estimation of linear models with unobserved effects on high volumes of data, which has never been feasible before. We are proud and honored to bridge the gap between the methods and technologies in econometrics and big data domains, utilizing their best capabilities.

Of course, our work could make additional contributions and be further extended. A natural extension of our implementation in TensorFlow would be to enable also the estimation of fixed effects models. The main challenge here arises from the fact that these models heavily rely on appropriate grouping and splitting of the data in order to perform the group-wise calculation for fitting and evaluation. Since such kind of preprocessing on very large datasets is not a core capability of TensorFlow, different techniques would be necessary compared to the way we are able to do this with Apache Spark. One promising approach to overcome this limitation is to utilize cutting-edge cloud technologies for data ingestion, such as Amazon Athena, as a complement to TensorFlow. This would provide the user not only with a highly efficient but also a cost-optimal solution.

Another natural extension of our work would be to implement a generalized fixed effects model that goes beyond just fitting a fixed effects logistic regression. Looking at the way we compute the gradient and Hessian in a distributed manner, the way we do Newton method iterations, and the way we compute the statistical summary, it becomes clear that most of the process (at least in the unconditional model) does not depend on a specific choice of link function. Thus, it should be relatively easy to generalize to any fixed effects generalized linear model by changing only a few lines of code. We have put this principle to the test by implementing a fixed effects Poisson regression within our framework that

got good results on large test datasets. Following on this path implementing, a fixed effects probit, Gamma, inverse-Gaussian and other regressions would be an interesting continuation of our project.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[2] J. Aldrich, J. Nelson, F. Nelson, E. Adler, and i. Sage Publications. *Linear Probability, Logit, and Probit Models*. Linear Probability, Logit and Probit Models. SAGE Publications, 1984.

[3] T. Amemiya. Qualitative response models: A survey. *Journal of Economic Literature*, 19(4):1483–1536, 1981.

[4] M. Bartholomew-Biggs, S. Brown, B. Christianson, and L. Dixon. Automatic differentiation of algorithms. *Journal of Computational and Applied Mathematics*, 124(1-2):171–190, 2000.

[5] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 18(1):5595–5637, 2017.

[6] N. Breslow. Covariance analysis of censored survival data. *Biometrics*, pages 89–99, 1974.

[7] A. C. Cameron and P. K. Trivedi. *Microeconometrics: Methods and Applications*. Cambridge University Press, 2005.

[8] G. Chamberlain. Analysis of covariance with qualitative data. *The Review of Economic Studies*, 47(1):225–238, 1980.

[9] E. A. CHAMBERS and D. R. COX. Discrimination between alternative binary response models. *Biometrika*, 54(3-4):573–578, 12 1967.

[10] S. Chapple, E. Troup, and T. Forster. *Mastering Parallel Programming with R*. Community experience distilled. Packt Publishing, 2016.

[11] R. Core-Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[12] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12(null):2121â“2159, July 2011.

[13] B. Efron. The efficiency of cox's likelihood function for censored data. *Journal of the American statistical Association*, 72(359):557–565, 1977.

[14] B. Efron and D. V. Hinkley. Assessing the accuracy of the maximum likelihood estimator: Observed versus expected Fisher information. *Biometrika*, 65(3):457–483, 12 1978.

[15] F. Eicker. Asymptotic normality and consistency of the least squares estimators for families of linear regressions. *The Annals of Mathematical Statistics*, pages 447–456, 1963.

[16] V. Farewel and R. L. Prentice. The approximation of partial likelihood with emphasis on case-control studies. *Biometrika*, 67(2):273–278, 1980.

[17] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[18] J. Hahn and W. Newey. Jackknife and analytical bias reduction for nonlinear panel models. *Econometrica*, 72(4):1295–1319, 2004.

[19] I. Hertz-Picciotto and B. Rockhill. Validity and efficiency of approximation methods for tied survival times in cox regression. *Biometrics*, pages 1151–1156, 1997.

[20] N. J. Higham. Cholesky factorization. *WIREs Computational Statistics*, 1(2):251–254, 2009.

[21] P. J. Huber et al. The behavior of maximum likelihood estimates under nonstandard conditions. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 221–233. University of California Press, 1967.

[22] J. Johnson. Rethinking floating point for deep learning. *arXiv preprint arXiv:1811.01721*, 2018.

[23] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[24] J. Neyman and E. L. Scott. Consistent estimates based on partially consistent observations. *Econometrica*, 16(1):1–32, 1948.

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[26] H. ren Fang and D. P. O'leary. Modified cholesky algorithms: A catalog with new approaches. *Math. Prog. A*, page 2008.

[27] A. Stammann, F. Heiß, and D. McFadden. Estimating fixed effects logit models with large panel data. Number G01-V3 in Beiträge zur Jahrestagung des Vereins für Socialpolitik 2016: Demographischer Wandel - Session: Microeconometrics, Kiel und Hamburg, 2016. ZBW - Deutsche Zentralbibliothek für Wirtschaftswissenschaften, Leibniz-Informationszentrum Wirtschaft.

[28] H. White. A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity. *Econometrica: journal of the Econometric Society*, pages 817–838, 1980.

[29] N. Whitehead and A. Florea. Floating point and ieee-754 compliance for nvidia gpus. *Nvidia Whitepaper*, 2017.

[30] J. M. Wooldridge. *Econometric analysis of cross section and panel data*. MIT press, 2010.

[31] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, Oct. 2016.

[32] A. Zeileis. Object-oriented computation of sandwich estimators. *Journal of Statistical Software, Articles*, 16(9):1–16, 2006.