TU Dortmund University
Faculty of Statistics

Data Reduction for Efficient Probit Regression

A Master's Thesis by Christian Peters

Dortmund, October 26, 2021

1. Supervisor: Dr. Alexander Munteanu
2. Supervisor: Prof. Dr. Katja Ickstadt

Contents

1 Introduction

Content.

2 The Probit Model

The probit model is a special case of the generalized linear model (GLM) described in [McCullagh and Nelder, 1989]. It is a statistical method for analyzing binary datasets, which we introduce in the following definition.

Definition 1 (Dataset)**.** *Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ be a set containing $n \in \mathbb{N}$ pairs of observations $x_i \in \mathbb{R}^d$, $y_i \in \{0, 1\}$. We call $\mathcal{D}$ a d-dimensional (binary) dataset.*

We can use this definition of a dataset (we will omit the term binary from now on since we will only be dealing with binary datasets in this work), to describe a whole range of possible scenarios that can be subjected to statistical analysis. For example, the x_i could represent some information of a patient, such as blood pressure or weight, and the y_i could indicate the presence or the absence of a heart disease.

In situations like this, we are often interested in modeling the relationship between the explanatory quantities x_i and the outcomes y_i. We need models, that can help us to answer questions about the data such as "Which factors increase/decrease the risk of suffering from a heart disease?", or "How likely is it, that a given patient will suffer from a heart disease?". The probit model is one of many approaches to model such a relationship in a probabilistic manner. It is described in detail in references like [McCullagh and Nelder, 1989], [Agresti, 2015] or [Fahrmeir et al., 2013].

We will outline the core assumptions of the probit model below, but instead of directly starting with its GLM formulation, we introduce it as a so-called latent variable model, which enables us to naturally arrive not only at its GLM specification, but also at a powerful sampling algorithm that enables us to efficiently apply the probit model in the realm of Bayesian data analysis.

2.1 Introduction as a Latent Variable Model

When using a probit model to analyze a d-dimensional dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, we implicitly make a set of assumptions about how the data was generated. Since it is reasonable to assume, that there is a degree of randomness involved in the data generating process, we model the y_i as realizations of independent random variables Y_i, which is the first assumption of the probit model.

The second assumption is that there is a hidden random quantity Y_i^* that is associated with each Y_i such that it directly determines its outcome:

$$Y_i = \begin{cases} 1, & \text{if } Y_i^* > 0 \\ 0, & \text{if } Y_i^* \leq 0 \end{cases} \tag{1}$$

The Y_i^* are also assumed to be independent from each other and, as already noted, unobservable, which is the reason why the Y_i^* are also called latent variables and why the probit model can also be thought of as a latent variable model.

The third and final assumption of the probit model defines the distribution of the Y_i^* and its part of the relationship between the non-random explanatory quantities x_i

and the outcomes y_i. In order to describe this relationship more concisely, we put all the observations x_i inside of a matrix $X \in \mathbb{R}^{n \times d}$ in such a way, that the i-th row of X corresponds to x_i. In the literature, this matrix X is often called the *model matrix* (see for example [Agresti, 2015]). We do the same with the Y_i^* and put them in a random vector Y^* as well, such that Y_i^* constitutes the i-th element of Y^*.

We are now ready for the third assumption of the probit model: The explanatory variables x_i influence Y_i^* in the form of a classical linear model:

$$Y^* = X\beta + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I), \tag{2}$$

where $\beta \in \mathbb{R}^d$ is the parameter vector of the linear model, ϵ is a normal distributed vector with independent components of mean zero and variance σ^2, and $I \in \mathbb{R}^{n \times n}$ is the $n \times n$ identity matrix. It follows directly that Y^* is also normal distributed: $Y^* \sim \mathcal{N}(X\beta, \sigma^2 I)$.

These three assumptions are already a complete specification of the probit model and are summarized in the following definition as a brief recapitulation:

Definition 2 (Probit Model)**.** *A d-dimensional binary dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ with model matrix $X \in \mathbb{R}^{n \times d}$ was generated by a probit model with parameters $\beta \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}_{>0}$, if the following three assumptions are true:*

1. *The observations $y_1, ..., y_n$ are realizations of independent binary random variables $Y_1, ..., Y_n$.*
2. *The outcomes of $Y_1, ..., Y_n$ are determined by hidden continuous random variables $Y_1^*, ..., Y_n^*$ by thresholding: If $Y_i^* > 0$, then $Y_i = 1$, and if $Y_i^* \leq 0$, then $Y_i = 0$.*
3. *The vector of hidden variables Y^* follows a multivariate normal distribution: $Y^* \sim \mathcal{N}(X\beta, \sigma^2 I)$, where $\beta \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}_{>0}$ are the model parameters.*

Based on this definition, it is straight forward to determine the distribution of the response variables Y_i. We can calculate the probability $P(Y_i = 1)$ like this:

$$P(Y_i = 1) = P(Y_i^* > 0) = 1 - P(Y_i^* \leq 0) = 1 - P\left(\frac{Y_i^* - x_i^T\beta}{\sigma} \leq -\frac{x_i^T\beta}{\sigma}\right) = \Phi\left(\frac{x_i^T\beta}{\sigma}\right),$$

where $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution:

$$\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2} dz.$$

The result $P(Y_i = 1) = \Phi\left(\frac{x_i^T\beta}{\sigma}\right)$ leads us to an interesting observation: Both parameters β and σ are unknown model parameters and every value of σ can be compensated by a corresponding scaling of β. This means that, because we can't observe the hidden variables Y_i^*, it is impossible to determine which β and which σ generated the data without any prior knowledge. We can only draw conclusions with regard to the scaled parameter $\frac{1}{\sigma}\beta$. In this situation, we say that β and σ are *not identifiable*.

For this reason, in literature like [Fahrmeir et al., 2013] or [Agresti, 2015], it is often argued, that without the loss of generality, we can assume that $\sigma = 1$ and arrive at

$$P(Y_i = 1) = \Phi(x_i^T \beta).$$

Conversely, since Y_i is binary, it follows that

$$P(Y_i = 0) = 1 - P(Y_i = 1) = 1 - \Phi(x_i^T \beta) = \Phi(-x_i^T \beta),$$

and we arrive at the model equations:

$$Y_i \sim Bin(1, \pi_i), \quad \pi_i = \Phi(x_i^T \beta), \tag{3}$$

where $Bin(1, \pi_i)$ is a Bernoulli distribution with success probability $\pi_i = \Phi(x_i^T \beta)$.

2.2 A Special Case of the Generalized Linear Model

The final equations of the probit model that we arrived at in equation 3 are a special case of a more general model concept, the generalized linear model (GLM), that we briefly touch on below.

Generalized linear models consist of three components. The first one is the so called *random component*, a set of $n \in \mathbb{N}$ independent random variables $\{Y_i\}_{i=1}^n$. In GLMs, the distribution of these random variables is assumed to be a member of the *exponential family*, a broad family of probability distributions that encompasses the normal distribution, the binomial distribution and many others. It is characterized in more detail in [Agresti, 2015].

The second component of a GLM is the *linear predictor*. Just like in the probit model, we also assume that we are presented with some fixed observations $\{x_i \in \mathbb{R}^d\}_{i=1}^n$, that are assumed to have some explanatory power with regard to the Y_i. We thus call these observations the explanatory quantities. The linear predictor is used to relate the explanatory quantities to the distribution of the Y_i by linearly combining them as follows:

$$\eta_i = x_i^T \beta,$$

where $\eta_i \in \mathbb{R}$ denotes the linear predictor related to observation x_i and $\beta \in \mathbb{R}^d$ is the unknown parameter vector of the GLM that has to be estimated when fitting the model.

The third component of a GLM is the so called *link function*. This is a monotonic, differentiable and invertible function g that connects the linear predictor η_i to the distribution of the Y_i like this:

$$g(E[Y_i]) = \eta_i.$$

We are thus using the link function g to transform the expected value $E[Y_i]$ in such a way that it can be predicted by a linear model, hence the name *generalized linear models*.

Equivalently, we can also characterize this relationsthip by using the inverse function $h = g^{-1}$, also called the *response function*:

$$E[Y_i] = h(\eta_i).$$

We are now ready to establish the connection between the probit model and the generalized linear model. As we saw in equation 3, the assumptions of the probit model imply that the Y_i follow independent binomial distributions with a success probability of $\pi_i = \Phi(x_i^T \beta)$. The binomial distribution is a member of the exponential family, so we can also think of the Y_i as the random component of a GLM.

It also follows directly from the binomial distribution that $E[Y_i] = \pi_i$, thus we have from the probit model equations that $\pi_i = E[Y_i] = \Phi(x_i^T \beta)$, and equivalently $\Phi^{-1}(E[Y_i]) = x_i^T \beta$. Thus, we can think of Φ as the response function of a GLM and Φ^{-1}, the quantile function of the standard normal distribution, as the link function. The function Φ^{-1} is also known as the *probit function*, hence the name probit model.

2.3 Parameter Estimation

The parameters of generalized linear models and therefore the parameters of the probit model are usually estimated by using the *maximum likelihood method*. This method seeks to maximize the likelihood that some observed dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ was generated under the assumptions of the model, given some parameter vector $\beta \in \mathbb{R}^d$.

To make notation a little easier, we also put the outcomes y_i in a vector $y \in \{0,1\}^n$ such that y_i is the i-th component of y. In the same way, we also put the random variables Y_i inside of a random vector Y.

In the probit model, the likelihood function for a dataset $\mathcal{D}$ is given by

$$\mathcal{L}_{\mathcal{D}}(\beta) = P(Y = y|\beta) = \prod_{i=1}^{n} P(Y_i = y_i|\beta), \tag{4}$$

because the Y_i are independent. By using a little trick, we can write $P(Y_i = y_i|\beta)$ as a single expression by combining the equations $P(Y_i = 1) = \Phi(x_i^T \beta)$ and $P(Y_i = 0) = \Phi(-x_i^T \beta)$ from section 2.1 like this:

$$P(Y_i = y_i|\beta) = \Phi[(2y_i - 1)x_i^T \beta],$$

which works because $2y_i - 1 = 1$ for $y_i = 1$ and $2y_i - 1 = -1$ for $y_i = 0$. This enables us to arrive at the likelihood

$$\mathcal{L}_{\mathcal{D}}(\beta) = \prod_{i=1}^{n} P(Y_i = y_i|\beta) = \prod_{i=1}^{n} \Phi[(2y_i - 1)x_i^T \beta] = \prod_{i=1}^{n} \Phi(-z_i^T \beta). \tag{5}$$

Here, we introduced the new vector $z_i = -(2y_i - 1)x_i$, which will simplify the notation later on.

The maximum likelihood estimate for β is then given by

$$\hat{\beta} \in \underset{\beta \in \mathbb{R}^d}{\operatorname{argmax}}\ \mathcal{L}_{\mathcal{D}}(\beta), \tag{6}$$

and for $n \to \infty$ it holds that $E[\hat{\beta}] = \beta$ [Fahrmeir et al., 2013].

However, for finite sample sizes, the existence of $\hat{\beta}$ cannot be guaranteed and is dependent on the observed data. An overview of the conditions for the existence and uniqueness of $\hat{\beta}$ is given in [Demidenko, 2001]. In particular, there is one important condition shown in [Lesaffre and Kaufmann, 1992], that is related to the concept of linear separability, which we introduce in the following definition.

Definition 3 (Linear separability)**.** *Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ be a d-dimensional binary dataset. Let $S_0 = \{i \in [n] : \ y_i = 0\}$ and $S_1 = \{i \in [n] : \ y_i = 1\}$. If there exists a $\beta \in \mathbb{R}^d \setminus \{0\}$ such that*

$$\forall i \in S_0 : \ x_i^T \beta \leq 0 \quad \textit{and} \quad \forall i \in S_1 : \ x_i^T \beta \geq 0,$$

then we call $\mathcal{D}$ linearly separable.

Intuitively speaking, a dataset is linearly separable if there exists a hyperplane that perfectly separates the datapoints labeled with 1 from the datapoints labeled with 0. This property of a dataset is a both sufficient and necessary condition for the existence of the maximum likelihood estimate $\hat{\beta}$, as stated in the following theorem.

Theorem 1 ([Lesaffre and Kaufmann, 1992])**.** *Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ be a d-dimensional binary dataset. The maximum likelihood estimate $\hat{\beta}$ for the parameter β of the probit model exists if and only if $\mathcal{D}$ is not linearly separable.*

In [Haberman, 1974], it was further shown, that if the maximum likelihood estimate exists and the model matrix X has full column rank, i.e. $rank(X) = d$, then it is also unique. It now remains to explore, how the maximum likelihood optimization problem can be solved in such a case.

2.3.1 Finding the Maximum Likelihood Estimate

For the reason that the likelihood function $\mathcal{L}_\mathcal{D}(\beta)$ is numerically inconvenient to maximize, the natural logarithm is often applied as a transformation to simplify the optimization problem:

$$\ell_\mathcal{D}(\beta) = \ln \mathcal{L}_\mathcal{D}(\beta) = \sum_{i=1}^n \ln \Phi(-z_i^T \beta). \tag{7}$$

Since we later wish to interpret $\ell_\mathcal{D}$ as a loss function, we prefer to minimize the negative value of $\ell_\mathcal{D}$ rather than maximizing:

$$f(\beta) = -\ell_\mathcal{D}(\beta) = \sum_{i=1}^n \ln \left(\frac{1}{1 - \Phi(z_i^T \beta)} \right) = \sum_{i=1}^n g(z_i^T \beta). \tag{8}$$

Here, we define $g(x) = \ln \left(\frac{1}{1-\Phi(x)} \right)$ and call it the *probit loss*, i.e. the loss-function that determines how much each z_i contributes to the total loss $f(\beta)$ for a given value of β.

At this point, we could already elaborate on the minimization of $f(\beta)$, but there is one more generalization that we have to make, which will later be needed when applying

the theory of data reduction to the probit model: We have to introduce positive sample weights $w_1, ...w_n$, alternatively specified by the weight vector $w \in \mathbb{R}^n_{>0}$, that give a positive weight to each datapoint in the objective function:

$$f_Z^w(\beta) = \sum_{i=1}^{n} w_i g(z_i^T \beta). \tag{9}$$

Here, we also introduced the subscript Z, which refers to the matrix $Z \in \mathbb{R}^{n \times d}$, where the i-th row of Z is given by z_i, but if Z and w are clear from the context, we will usually omit it and simply refer to f_Z^w by f. In the rest of this work, we will be referring to Z as the *scaled model matrix*.

The optimization of f is usually done by applying the Newton-Raphson algorithm, an iterative procedure that starts at some initial guess $\beta^{(0)}$ and successively updates it like this:

$$\beta^{(t)} = \beta^{(t-1)} - \left(\frac{\partial^2 f(\beta^{(t-1)})}{\partial \beta \partial \beta^T} \right)^{-1} \cdot \frac{\partial f(\beta^{(t-1)})}{\partial \beta}, \tag{10}$$

where $\left(\frac{\partial^2 f(\beta^{(t-1)})}{\partial \beta \partial \beta^T} \right)^{-1}$ refers to the inverse of the hessian matrix of f, evaluated at $\beta^{(t-1)}$, and $\frac{\partial f(\beta^{(t-1)})}{\partial \beta}$ refers to the gradient of f, evaluated at $\beta^{(t-1)}$. The idea behind this procedure is, broadly speaking, to approximate f locally around $\beta^{(t)}$ as a second degree taylor-polynomial and then analytically find the minimum of this polynomial. The minimum of this local polynomial approximation of f is then iteratively used as the basis for the next step of the Newton-Raphson algorithm.

It remains to find the gradient as well as the hessian matrix of f. Because f is a sum of the function g evaluated at different points, it makes sense to first determine the derivative of g. This can be accomplished by using the chain rule as follows:

$$\begin{aligned}
\frac{d}{dx} g(x) &= \frac{d}{dx} \ln \left(\frac{1}{1 - \Phi(x)} \right) \\
&= (1 - \Phi(x)) \cdot \frac{d}{dx} \left(\frac{1}{1 - \Phi(x)} \right) \\
&= (1 - \Phi(x)) \cdot \frac{(-1)}{(1 - \Phi(x))^2} \cdot \frac{d}{dx} (1 - \Phi(x)) \\
&= \frac{(-1)}{1 - \Phi(x)} \cdot (-1) \cdot \phi(x) \\
&= \frac{\phi(x)}{1 - \Phi(x)},
\end{aligned} \tag{11}$$

where $\phi(x)$ is the density function of the standard normal distribution function:

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}.$$

We can use this result to calculate the gradient of f:

$$\begin{aligned}\frac{\partial}{\partial\beta}f(\beta) &= \frac{\partial}{\partial\beta}\sum_{i=1}^{n} w_i g(z_i^T\beta) \\ &= \sum_{i=1}^{n} w_i z_i g'(z_i^T\beta) \\ &= \sum_{i=1}^{n} w_i z_i \frac{\phi(z_i^T\beta)}{1-\Phi(z_i^T\beta)}\end{aligned} \tag{12}$$

Next, we need to determine the hessian matrix of f. In order to do this, we again start by finding the second derivative of g, this time using the quotient rule:

$$\begin{aligned}\frac{d^2}{dx^2}g(x) &= \frac{d}{dx}\frac{\phi(x)}{1-\Phi(x)} \\ &= \frac{\phi'(x)(1-\Phi(x)) - \phi(x)\cdot(-1)\cdot\phi(x)}{(1-\Phi(x))^2} \\ &= \frac{(-1)\cdot x\cdot\phi(x)(1-\Phi(x)) - \phi(x)\cdot(-1)\cdot\phi(x)}{(1-\Phi(x))^2} \\ &= \frac{[\phi(x)]^2 - x\cdot\phi(x)\cdot(1-\Phi(x))}{(1-\Phi(x))^2} \\ &= \left(\frac{\phi(x)}{1-\Phi(x)}\right)^2 - x\cdot\frac{\phi(x)}{1-\Phi(x)} \\ &= \frac{\phi(x)}{1-\Phi(x)}\left(\frac{\phi(x)}{1-\Phi(x)} - x\right) \\ &= g'(x)\cdot(g'(x)-x)\end{aligned} \tag{13}$$

We can now use this result to find the hessian matrix of f:

$$\begin{aligned}\frac{\partial^2}{\partial\beta\partial\beta^T}f(\beta) &= \sum_{i=1}^{n}\frac{\partial^2}{\partial\beta\partial\beta^T}w_i g(z_i^T\beta) \\ &= \sum_{i=1}^{n} w_i z_i z_i^T g'(z_i^T\beta)(g'(z_i^T\beta) - z_i^T\beta) \\ &= \sum_{i=1}^{n} w_i z_i z_i^T \frac{\phi(z_i^T\beta)}{1-\Phi(z_i^T\beta)}\left(\frac{\phi(z_i^T\beta)}{1-\Phi(z_i^T\beta)} - z_i^T\beta\right).\end{aligned} \tag{14}$$

Because it can be shown, that $f(\beta)$ is a convex function [Wedderburn, 1976], and that the Newton Raphson algorithm converges to a global optimum when applied to a convex function [Nocedal and Wright, 2006], the optimization procedure converges to the maximum likelihood estimate $\hat{\beta}$ under the condition that the data is not linearly separable.

2.4 The Bayesian Perspective

The most fundamental difference between the bayesian approach to the probit model and the frequentist approach that was discussed above, is the assumption, that the model parameter β is not a fixed value, but a random variable with a probability distribution. The goal of bayesian data analysis is to draw conclusions about the distribution of the model parameter and to update these conclusions after observing more and more data.

A detailed overview of the principles of bayesian data analysis would certainly go beyond the scope of this work, but the interested reader will find a comprehensive reference in [Gelman et al., 2013]. In this section, we will merely touch on the most essential concepts, which are required in order to understand the bayesian view on the probit model.

2.4.1 Prior and Posterior Distributions

To characterize the prior uncertainty about the model parameter β, the first step of bayesian data analysis is to specify a so called *prior distribution*. In the probit model, one common choice that is also described in [Fahrmeir et al., 2013] is to assume that

$$\beta \sim \mathcal{N}(\mu_\beta, \Sigma_\beta), \tag{15}$$

i.e. β follows a normal distribution with mean $\mu_\beta \in \mathbb{R}^d$ and covariance matrix $\Sigma_\beta \in \mathbb{R}^{d\times d}$.

We can think of μ_β and Σ_β as a way to include prior knowledge into the model. If such knowledge is not present, we can choose μ_β and Σ_β in a more general fashion, perhaps we decide to set $\mu_\beta = 0$ and $\Sigma_\beta = \sigma_\beta \cdot I$ for a large value of σ_β, which would be an example of an *uninformative* prior because of the relatively unrestrictive assumptions. Alternatively, we could even go as far and also specify prior distributions on μ_β and Σ_β, which would lead us into the realm of hierarchical models (see [Gelman et al., 2013] for more details). But this would definitely go beyond the scope of this work, which is why we assume from now on that the values of μ_β and Σ_β are specified beforehand in a reasonable manner.

The next step in the process of bayesian data analysis is to determine how we should update our initial prior assumptions about β after we observed some new data represented by the dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$. Ultimately, the goal is to determine the *posterior distribution* of β given the new data, represented by the probability density function $p(\beta|Y = y)$, where y is the vector of observations and Y is the random vector that we assumed to have generated these observations in the probit model.

We can find the posterior distribution by making use of the bayes rule:

$$p(\beta|Y = y) = \frac{p(Y = y|\beta)p(\beta)}{p(Y = y)}. \tag{16}$$

This relationship tells us, that in order to arrive at the posterior distribution, there are three different parts that we have to combine.

The first part is the likelihood function $p(Y = y|\beta)$, which we already dealt with in section 2.3, and the second part is the prior density function $p(\beta)$, that we assumed to be normal.

The third and most challenging part to compute is the quantity $p(Y = y)$. We can see why it is so challenging by writing it out:

$$\begin{aligned} p(Y = y) &= \int p(Y = y|\beta)p(\beta)d\beta \\ &= \int \prod_{i=1}^{n} \Phi(-z_i^T\beta) \frac{1}{\sqrt{(2\pi)^d \det \Sigma_\beta}} \exp\left(-\frac{1}{2}(\beta - \mu_\beta)^T \Sigma_\beta^{-1}(\beta - \mu_\beta)\right) d\beta \end{aligned} \tag{17}$$

This infinite integral over all possible values of β is impossible to solve analytically, which means that it's impossible to exactly compute the posterior distribution for the probit model. But luckily, encountering an intractable integral like this is quite common in bayesian data analysis, so there are workarounds that still allow us to analyze the posterior distribution, even though we are unable to determine it exactly.

The first consideration is, that we could also analyze the posterior distribution if we had a large enough sample of it available instead. When the sample size is big enough, the Glivenko-Cantelli theorem tells us that the empirical posterior distribution converges to the true posterior distribution [Vaart, 1998]. This allows us to analyze the posterior distribution by analyzing a large enough sample of it, but the problem of how to obtain such a sample still remains.

In practice, instead of directly sampling from $p(\beta|Y = y)$, the posterior distribution can be approximated by so called Markov chain Monte Carlo (MCMC) methods. One such method that works particularly well for the probit model is the Gibbs sampler, which is described in the next section.

2.4.2 Gibbs Sampling in the Probit Model

Gibbs sampling is an iterative tool for drawing samples from probability distributions, that was first applied in the context of bayesian inference by [Gelfand and Smith, 1990] and has been adapted to the probit model by [Albert and Chib, 1993] using the idea of *data augmentation*, which was first introduced in [Tanner and Wong, 1987]. We describe this idea in the following section, as it yields an efficient algorithm for sampling from the posterior distribution of the probit model.

Remember, that the probit model has the following components: The vector of latent variables Y^* that follows a linear model $Y^* \mid \beta \sim \mathcal{N}(X\beta, 1)$, where we assume that $\sigma = 1$ for reasons of identifiability (see section 2.1) and the random vector Y that produces the observed outcomes y by thresholding: If $Y_i^* > 0$, then $Y_i = 1$ and $Y_i = 0$ otherwise. We also assumed a normal prior distribution: $\beta \sim \mathcal{N}(\mu_\beta, \Sigma_\beta)$.

Now, imagine that we knew the outcomes of the latent variable vector Y^*. The conditional distribution of β given the realization y^* of the latent variables can be shown to be normal [Albert and Chib, 1993]:

$$\beta \mid Y^* = y^* \ \sim \ \mathcal{N}(b, B), \tag{18}$$

where $b = (\Sigma_\beta^{-1} + X^TX)^{-1}(\Sigma_\beta^{-1}\mu_\beta + X^Ty^*)$ and $B = (\Sigma_\beta^{-1} + X^TX)^{-1}$. From this distribution, it is possible to sample efficiently.

The problem is, that in reality we can't observe the latent variables and therefore we don't know the realizations y^*. Here, an important finding by [Albert and Chib, 1993] comes into play: If we could observe β and see the realization $\tilde{\beta}$, then we could determine the conditional distribution of the latent variable vector Y^*:

$$Y_i^* \mid \beta = \tilde{\beta},\ Y_i = y_i \ \sim \begin{cases} \mathcal{N}(x_i^T\tilde{\beta}, 1) \text{ truncated at the left by 0}, & \text{if } y_i = 1 \\ \mathcal{N}(x_i^T\tilde{\beta}, 1) \text{ truncated at the right by 0}, & \text{if } y_i = 0 \end{cases} \tag{19}$$

This means, that given a realization $\tilde{\beta}$ and the observed values in y, the latent variables follow a truncated normal distribution, from which it is also possible to sample efficiently.

These two observations bring us directly to the Gibbs sampling algorithm for the probit model. The first step of this procedure is to determine a starting value $\tilde{\beta}^{(0)}$. [Albert and Chib, 1993] suggest that this could for example be the maximum likelihood estimate, that we already discussed in section 2.3.

The next step of the Gibbs sampling algorithm is to use this value $\tilde{\beta}^{(0)}$ to sample a realization $y^{*(1)}$ from the latent variable vector Y^*, by using the conditional distribution in equation 19. Given $y^{*(1)}$, it is then possible to sample a new value $\tilde{\beta}^{(1)}$ from the normal distribution in equation 18, which starts a new cycle. These two sampling steps, which can both be carried out efficiently, are repeated until the desired amount of samples is reached. See Algorithm 1 for the full algorithm.

To sum up, we augmented the observed data by incorporating the hidden variables Y^* to arrive at a two-stage procedure that draws alternating samples from the conditional distributions of β and Y^*, hence the name data augmentation.

Algorithm 1: Gibbs Sampler for the Probit Model

Input: Dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ with model matrix X, prior mean $m \in \mathbb{R}^d$, prior covariance matrix $M \in \mathbb{R}^{d\times d}$, sample size $k \in \mathbb{N}$

Output: A sample $\beta_1, ..., \beta_k$ from the posterior distribution

Set $B = (M^{-1} + X^TX)^{-1}$
Initialize $\beta_0 = \hat{\beta}$, where $\hat{\beta}$ is the MLE for β computed on $\mathcal{D}$
for $j = 1, ..., k$ **do**
  for $i = 1, ..., n$ **do**
    if $y_i = 1$ **then**
      Sample $y_i^{*(j)}$ from $\mathcal{N}(x_i^T\beta_{j-1}, 1)$ truncated at the left by 0
    else if $y_i = 0$ **then**
      Sample $y_i^{*(j)}$ from $\mathcal{N}(x_i^T\beta_{j-1}, 1)$ truncated at the right by 0
  Set $y^{*(j)} = \left(y_1^{*(j)}, ..., y_n^{*(j)}\right)^T$
  Set $b^{(j)} = B\left(M^{-1}m + X^Ty^{*(j)}\right)$
  Sample β_j from $\mathcal{N}\left(b^{(j)}, B\right)$
return $\beta_1, ..., \beta_k$

3 Coresets and Sensitivity Sampling

The Newton-Raphson algorithm for optimizing the objective function of the probit model, as well as the Gibbs sampler, are reasonably efficient when the datasets are of small to moderate size. Usually, this is the case when it's possible to store the model matrix X into the main memory. But problems arise, when the datasets are getting so big, that this is no longer possible. What should we do in such a case?

One idea to deal with this issue is that we could select a smaller subset $\mathcal{C}$ of our initial dataset $\mathcal{D}$, that represents the characteristics of the original data well in some sense. We hope, that when we execute the computationally expensive optimization algorithms on the smaller subset, we still get similar results as if we executed the algorithms on the original dataset. But what does it mean for a subset $\mathcal{C}$ to be representative of $\mathcal{D}$? And how could we come up with an algorithm that selects such a subset efficiently? The method of *coresets* (see for example [Munteanu and Schwiegelshohn, 2018]) is one way of dealing with these questions, which we will explore in this chapter.

So, what is a coreset? As the name suggests, we are talking about a subset $\mathcal{C} \subseteq \mathcal{D}$ of our initial dataset, that fulfills some very special requirements which ensure that the original dataset is well represented for our problem. To be more specific, we want the objective function $f(\beta)$ evaluated on the coreset to be as close to the objective function on the original dataset as possible, for all $\beta \in \mathbb{R}^d$. Mathematically speaking, what we are interested in is a so-called $(1 \pm \epsilon)$ approximation of the objective function on the original dataset.

To understand what is meant by that, assume for a moment that we are given a function $f(\beta)$ and an approximation $\tilde{f}(\beta)$. If $\tilde{f}(\beta)$ is a $(1 \pm \epsilon)$ approximation of $f(\beta)$, it will never deviate from $f(\beta)$ more than a factor $(1 \pm \epsilon)$, i.e. we have for all $\beta \in \mathbb{R}^d$, that:

$$(1 - \epsilon) f(\beta) \leq \tilde{f}(\beta) \leq (1 + \epsilon) f(\beta).$$

This kind of approximation would then allow us to run an optimization algorithm on $\tilde{f}(\beta)$ and guarantee that our solution is close to the optimal solution on $f(\beta)$.

As we already hinted at, a coreset is simply a subset $\mathcal{C} \subseteq \mathcal{D}$ of our original dataset, that provides us with a $(1 \pm \epsilon)$ approximation of the original loss function. We formalize this concept in the following definition.

Definition 4 (Coreset)**.** *Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ be a d-dimensional dataset with scaled model matrix $Z \in \mathbb{R}^{n \times d}$, i.e. $z_i = -(2y_i - 1)x_i$ constitutes the i-th row of Z, and let $w \in \mathbb{R}^n_{>0}$ be a vector of positive sample weights. Let $\mathcal{C} \subseteq \mathcal{D}$ be a subset of $\mathcal{D}$ of size $|\mathcal{C}| = k$ with scaled model matrix $C \in \mathbb{R}^{k \times d}$ and a vector of positive sample weights $u \in \mathbb{R}^k_{>0}$. Let $\epsilon > 0$. We call $\mathcal{C}$ a $(1 \pm \epsilon)$-coreset of $\mathcal{D}$ for probit regression, if*

$$(1 - \epsilon) f_Z^w(\beta) \leq f_C^u(\beta) \leq (1 + \epsilon) f_Z^w(\beta) \quad \forall \beta \in \mathbb{R}^d,$$

where $f_Z^w(\beta) = \sum_{i=1}^n w_i g(z_i^T \beta)$ is the weighted objective function of the probit model and $g(x) = \ln\left(\frac{1}{1-\Phi(x)}\right)$ is the probit loss.

One thing to note here is that we do not only need to select the subset $\mathcal{C} \subseteq \mathcal{D}$, but we also have to come up with some new sample weights u. Intuitively speaking, this makes sense because when reducing the amount of data points in the objective function, which is achieved by selecting the subset $\mathcal{C}$, we are also naturally lowering its overall value, since g, the probit loss, is a positive function. The reweighting by u accounts for that, so we can still get a $(1 \pm \epsilon)$ approximation of the original loss function.

Let us now consider perhaps the most important aspect of this definition, which will determine the usefullness of *any* work in the domain of coresets: The coreset size $k = |\mathcal{C}|$. It can easily be verified, that we can always come up with a coreset when $k = n$, i.e. the so-called trivial coreset, where we simply select $\mathcal{C} = \mathcal{D}$. But such a coreset doesn't help us at all with our goal of reducing the computational burden of the optimization and Gibbs sampling algorithms. Informally speaking, we want k to be small. But how small is small enough? Usually, we can consider it a success, if we can find coresets where $k \in O(\log(n))$, using the big-o notation to indicate that k is not much larger than the logarithm of the amount of data points. If, for example, we had a dataset with one billion observations, i.e. $n = 1,000,000,000$, the natural logarithm of n would equal to roughly 20 datapoints. That sounds like a decent compression, doesn't it?

In the remainder of this work, we will refer to a coreset as small, if k is roughly logarithmic in n. Our goal is to construct algorithms, which will enable us to find such small coresets in the context of probit regression.

3.1 Do small coresets always exist?

Before attempting to construct an algorithm that is able to find small coresets, we first have to investigate if such a goal is even attainable, i.e. we have to make sure that small coresets even exist.

Without imposing any restrictions on the datasets, it turns out that it is not difficult to find a counter example, i.e. to find a dataset $\mathcal{D}$, such that no subset $\mathcal{C} \subseteq \mathcal{D}$ of roughly logarithmic size can be a coreset. This negative result has first been proven in the context of logistic regression by the authors of [Munteanu et al., 2018], but it turns out that the problematic dataset that admits no small coresets is the same for probit regression as well.

This finding forces us to make a decision: Do we have to give up our search for small coresets because we now know that they don't always exist? Or is there still hope, perhaps by imposing some (very reasonable) restrictions on the class of datasets that we consider? Before we can turn to this discussion, we first reproduce the counter example for the general case in the following theorem.

Theorem 2. *There exists a dataset $\mathcal{D}$ of size $|\mathcal{D}| = n$, such that any $(1 \pm \epsilon)$-coreset $\mathcal{C}$ of $\mathcal{D}$ for probit regression has a size $k = |\mathcal{C}|$ of at least $k \in \Omega\left(\frac{n}{\log n}\right)$.*

Proof. We can construct such a dataset by showing how coresets can be used in a communication protocol for the so called INDEX game, a communication game for two players, Alice and Bob, which works like this:

Alice is given a random binary string $m \in \{0,1\}^n$ of n bits and Bob is given an index $i \in [n]$. The objective of the game is for Alice to send a message to Bob that allows Bob to obtain the value m_i of Alice's binary string m. It was shown in [Kremer et al., 1999], that the minimum length of a message sent by Alice that still allows Bob to obtain m_i with constant probability is in $\Omega(n)$ bits. We will now see, how a coreset for probit regression can be used to encode such a message.

The first step is for Alice to convert her binary string m into a dataset $\mathcal{D}$ as follows: For each entry m_j of her binary string where $m_j = 1$, she adds a point

$$x_j = \left(\cos\left(2\pi\frac{j}{n}\right), \sin\left(2\pi\frac{j}{n}\right), 1\right)^T$$

to her set $\mathcal{D}$ and labels it with $y_j = 1$, ending up with the dataset

$$\mathcal{D} = \{(x_j, 1)\}_{j \in \{i \in [n]:\ m_i = 1\}},$$

with all points being on the unit circle.

The next step for her is to construct a $(1 \pm \epsilon)$-coreset $\mathcal{C}$ of $\mathcal{D}$ for probit regression with sample weights $u \in \mathbb{R}^k_{>0}$ and to transmit both the coreset and the weight vector to Bob, which requires $O(\log(n))$ space for each point and weight. We will later see, how large the size $|\mathcal{C}| = k$ of this coreset must be, so that Bob can still obtain the value of m_i with constant probability.

As soon as Alice's coreset $\mathcal{C}$ arrives at Bob, Bob can use it to obtain the value of m_i. To do this, Bob first adds two new points

$$q_1 = \left(\cos\left(2\pi\frac{i-0.5}{n}\right), \sin\left(2\pi\frac{i-0.5}{n}\right), 1\right)^T$$

and

$$q_2 = \left(\cos\left(2\pi\frac{i+0.5}{n}\right), \sin\left(2\pi\frac{i+0.5}{n}\right), 1\right)^T$$

to the set and labels both points with 0 (see figure 1), i.e. Bob now has the dataset

$$\mathcal{C}' = \mathcal{C} \cup \{(q_1, 0)\} \cup \{(q_2, 0)\}.$$

Next, he uses this new dataset $\mathcal{C}'$ with scaled model matrix C' to minimize the weighted objective function $f^u_{C'}$ of the probit model, by using the Newton-Raphson optimization algorithm.

Taking a look at figure 1, it becomes evident, that Bobs points q_1 and q_2 are linearly separable from the other points if and only if Alice didn't add a point x_i, i.e. if $m_i = 0$. He can use the results of the optimization procedure to make a distinction between the two cases, which then allows him to determine the value of m_i like this:

In the case of $m_i = 1$, Bobs points are not linearly separable from Alices original points, which means that there must occur at least one misclassification at a cost of $g(0) = \log(2)$ for the original loss function. Because Bobs dataset $\mathcal{C}'$ allows him to

obtain a $(1 \pm \epsilon)$-approximation of the original cost function, he can check if the Newton-Raphson algorithm converges to a cost of at least $(1 - \epsilon)\log(2)$. In this case, he knows that Alice must have added the point x_i, which means that $m_i = 1$.

Conversely, if at any point during the optimization procedure the cost function drops below $(1 - \epsilon)\log(2)$ and approaches zero, Bob knows that Alice didn't add the point x_i, because his dataset $\mathcal{C}'$ is linearly separable. This will allow him to conclude that $m_i = 0$.

Let us now see, how large the size k of Alice's coreset must be for this protocol to work with constant probability. In [Kremer et al., 1999] it was shown, that the minimum length of a message that Alice must send in order for the protocol to work is in $\Omega(n)$ bits. Since each of the points that Alice created can be encoded in $\log(n)$ space, it follows from the lower bound that $\Omega(n) \subseteq \Omega(k\log(n))$, so k must be in $\Omega\left(\frac{n}{\log(n)}\right)$.

We can conclude, that if there existed a $(1 + \epsilon)$-coreset of Alice's dataset $\mathcal{D}$ for probit regression with size $k \in o\left(\frac{n}{\log(n)}\right)$, it would contradict the minimum message length of the INDEX communication game, which proves the theorem. □

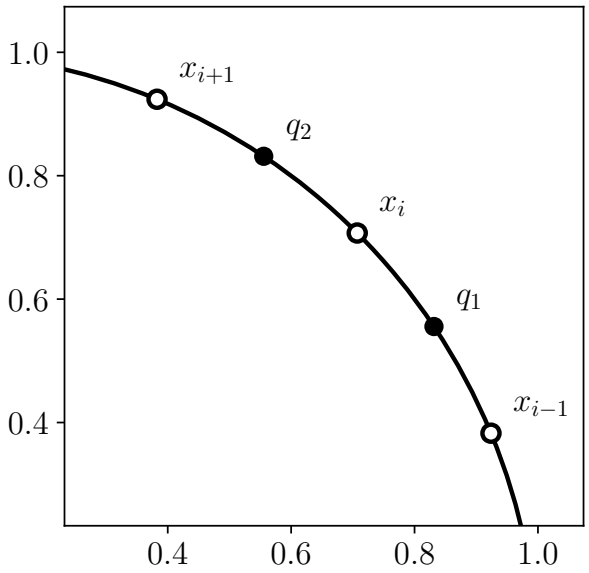

Figure 1: Bob places two points q_1 and q_2 in such a way on the unit circle, that they can be linearly seperated from the other points if and only if Alice didn't place a point at x_i.

We now have an example of a dataset for which no small coresets exist, which implies that in the general case, without any restrictions, there are no guarantees that it's even possible to find a small coreset. But there is one thing that we have to note: The counter example from the INDEX proof is by no means a dataset that could ever be reasonably subjected to a probit analysis. It consists of only positive labels! Further, it is easy to recognize, that the counter example is linearly separable. As we already saw in section 2.3, when estimating the model parameters, the maximum likelihood estimate only exists and is unique, when the data is not linearly separable. So yes, we found an

example dataset for which no small coresets exist, but does that mean that this particular "degenerate" example is relevant to the attainment of our goal of constructing efficient data reduction algorithms for the purpose of probit regression? Since the maximum likelihood extimate doesn't even exist for this dataset, it can be doubted, to say the least.

It therefore seems reasonable to impose some restrictions on the datasets under study. Since we are exclusively dealing with probit regression, it makes sense to restrict the class of data sets to those, where a probit model can at least be properly estimated, i.e. where the data is not linearly separable and were the maximum likelihood estimate exists and is unique.

The authors of [Munteanu et al., 2018] were dealing with similar issues in the context of logistic regression, so they decided to introduced a measure, which they call μ, that describes the degree of separability of a dataset. In their work, they were able to not only use this measure to restrict the class of datasets under study, which they defined as μ-complex, but also to relate the size of their coresets directly to μ. We will go down a similar path in the search of small coresets for probit regression, so the first step for us is to slightly adapt this measure for our purposes:

Definition 5. *(μ-complexity) Let $\mathcal{D}$ be a d-dimensional dataset of size $|\mathcal{D}| = n$ with scaled model matrix $Z \in \mathbb{R}^{n\times d}$, where $z_i \in \mathbb{R}^d$ constitutes the i-th row of Z and let $w \in \mathbb{R}^n_{>0}$ be a vector of positive weights. Let $I^+_\beta = \{i \in [n] : \ w_i z_i^T \beta > 0\}$ and let $I^-_\beta = \{i \in [n] : \ w_i z_i^T \beta < 0\}$. Let*

$$\mu_w(\mathcal{D}) = \sup_{\beta\in\mathbb{R}^d\setminus\{0\}} \frac{\sum_{i\in I^+_\beta} w_i (z_i^T\beta)^2}{\sum_{i\in I^-_\beta} w_i (z_i^T\beta)^2}.$$

We call the dataset $\mathcal{D}$ with weight vector w μ-complex, if there exists a $\mu \in \mathbb{R}$, such that $\mu_w(\mathcal{D}) \leq \mu$.

In order to understand, what exactly this measure does, we have to remember that the parameter vector β that is estimated in the probit model is the orthogonal vector of a hyperplane that separates the space of data points into two partitions. When using the probit model for binary classification, the decision to classify a point as positive or negative is often based on which side of the hyperplane the point is located. If it is possible to perfectly classify each point such that no errors are made, the dataset is called linearly separable. If it isn't linearly separable, there are always two subsets of datapoints: Those points that are correcly classified and those that are not. It turns out, that these two subsets are exactly represented by the sets I^+_β and I^-_β in the definition of μ. If we assume without losing generality, that for a given $\beta \in \mathbb{R}^d$, we classify a point as positive if $x_i^T\beta < 0$, i.e. it is on the opposite side of the hyperplane where the normal vector β points to, then $z_i^T\beta = -(2y_i - 1)x_i^T\beta > 0$ and I^+_β contains exactly all the indices of correctly classified points. Conversely, I^-_β then contains all the indices of incorrectly classified points. It is thus easy to see, that if every point is correctly classified, i.e. the dataset is linearly separable, then $I^-_\beta = \emptyset$ and $\mu = \infty$, so the dataset is not μ-complex.

The relationship between linear separability and μ-complexity is even stronger though. In the next theorem, we will show that a finite μ, i.e. the μ-complexity of a dataset, is exactly equivalent to linear separability.

Theorem 3. *Let $\mathcal{D}$ be a d-dimensional dataset of size $|\mathcal{D}| = n$ like in definition 5 (μ-complexity) and let $w \in \mathbb{R}^n_{>0}$ be a vector of positive weights. Then, the dataset $\mathcal{D}$ with weight vector w is μ-complex if and only if $\mathcal{D}$ is not linearly separable.*

Proof. We first prove the "$\Rightarrow$" direction, i.e. we show that if $\mathcal{D}$ is μ-complex, then it is not linearly separable. We do this by proving the equivalent contraposition that if $\mathcal{D}$ is linearly separable, then it is not μ-complex.

Let $S_0 = \{i \in [n] : \ y_i = 0\}$ and $S_1 = \{i \in [n] : \ y_i = 1\}$ like in definition 3 (linear separability). If $\mathcal{D}$ is linearly separable, then there exists a $\beta \in \mathbb{R}^d \setminus \{0\}$, such that

$$
\begin{aligned}
&\forall i \in S_0 : \ x_i^T\beta \geq 0 \quad \text{and} \quad \forall i \in S_1 : \ x_i^T\beta \leq 0 \\
\iff \\
&\forall i \in S_0 : \ (-1)x_i^T\beta \leq 0 \quad \text{and} \quad \forall i \in S_1 : \ x_i^T\beta \leq 0 \\
\iff \\
&\forall i \in S_0 : \ (2y_i - 1)x_i^T\beta \leq 0 \quad \text{and} \quad \forall i \in S_1 : \ (2y_i - 1)x_i^T\beta \leq 0 \\
\iff \\
&\forall i \in S_0 : \ -(2y_i - 1)x_i^T\beta \geq 0 \quad \text{and} \quad \forall i \in S_1 : \ -(2y_i - 1)x_i^T\beta \geq 0 \\
\iff \\
&\forall i \in S_0 : \ z_i^T\beta \geq 0 \quad \text{and} \quad \forall i \in S_1 : \ z_i^T\beta \geq 0 \\
\iff \\
&\forall i \in [n] : \ z_i^T\beta \geq 0 \\
\iff \\
&I_\beta^- = \{i \in [n] : \ w_i z_i^T\beta < 0\} = \emptyset \\
\iff \\
&\sum_{i \in I_\beta^-} w_i (z_i^T\beta)^2 = 0 \\
\Rightarrow \\
&\mu_w(\mathcal{D}) \geq \frac{\sum_{i \in I_\beta^+} w_i (z_i^T\beta)^2}{\sum_{i \in I_\beta^-} w_i (z_i^T\beta)^2} = \infty,
\end{aligned}
$$

which means that $\mathcal{D}$ is not μ-complex.

It now remains to prove the "$\Leftarrow$" direction, i.e. to show that if $\mathcal{D}$ is not linearly separable, then it is μ-complex. Again, we do this by proving the equivalent contraposition that if $\mathcal{D}$ is not μ-complex, then it is linearly separable.

The first step in order to do so is to show that we can restrict the supremum in $\mu_w(\mathcal{D})$

to finite β with $\|\beta\| = 1$:

$$\begin{aligned}
\mu_w(\mathcal{D}) &= \sup_{\beta \in \mathbb{R}^d \setminus \{0\}} \frac{\sum_{i \in I_\beta^+} w_i (z_i^T \beta)^2}{\sum_{i \in I_\beta^-} w_i (z_i^T \beta)^2} \\
&= \sup_{\beta \in \mathbb{R}^d \setminus \{0\}} \frac{\sum_{i \in I_\beta^+} \frac{1}{\|\beta\|^2} w_i (z_i^T \beta)^2}{\sum_{i \in I_\beta^-} \frac{1}{\|\beta\|^2} w_i (z_i^T \beta)^2} \\
&= \sup_{\beta \in \mathbb{R}^d \setminus \{0\}} \frac{\sum_{i \in I_\beta^+} w_i \left(z_i^T \frac{\beta}{\|\beta\|} \right)^2}{\sum_{i \in I_\beta^-} w_i \left(z_i^T \frac{\beta}{\|\beta\|} \right)^2} \\
&= \sup_{\tilde{\beta} \in \mathbb{R}^d,\ \|\tilde{\beta}\| = 1} \frac{\sum_{i \in I_\beta^+} w_i \left(z_i^T \tilde{\beta} \right)^2}{\sum_{i \in I_\beta^-} w_i \left(z_i^T \tilde{\beta} \right)^2},
\end{aligned}$$

which lets us conclude that even in the supremum, both expressions $\sum_{i \in I_\beta^+} w_i (z_i^T \beta)^2$ and $\sum_{i \in I_\beta^-} w_i (z_i^T \beta)^2$ are finite. This means that if $\mathcal{D}$ is not μ-complex, then the denominator must be zero, i.e. it must hold that there exists a $\beta \in \mathbb{R}^d \setminus \{0\}$ such that

$$\sum_{i \in I_\beta^-} w_i (z_i^T \beta)^2 = 0.$$

From here, we can follow the same chain of equivalences that we showed when proving the "⇒"-direction of the theorem, which leads us directly to the fact, that $\mathcal{D}$ in this case must be linearly separable, which concludes the proof. □

Having established the relationship between μ-complexity and linear separability, it directly follows that μ-complexity is also equivalent to the existence and uniqueness of the maximum likelihood estimate of the probit model, that we discussed in section 2.3.

From now on, we will subject our studies of coresets only to those datasets, that are μ-complex, i.e. not linearly separable and with existing and unique maximum likelihood estimate for the probit model.

We conclude this section by proving some simple inequalities regarding μ, which will later be helpful when constructing the coresets.

Lemma 1. *Let $\mathcal{D}$ be a d-dimensional and μ-complex dataset of size $|\mathcal{D}| = n$ with scaled model matrix $Z \in \mathbb{R}^{n \times d}$ and weight vector $w \in \mathbb{R}^n_{>0}$ like in definition 5. The following relationship holds for all $\beta \in \mathbb{R}^d$:*

$$\mu^{-1} \sum_{i \in I_\beta^-} w_i (z_i^T \beta)^2 \leq \sum_{i \in I_\beta^+} w_i (z_i^T \beta)^2 \leq \mu \sum_{i \in I_\beta^-} w_i (z_i^T \beta)^2.$$

Proof. If $\mathcal{D}$ with weights w is μ-complex, then

$$\begin{aligned} &\frac{\sum_{i \in I_\beta^+} w_i (z_i^T \beta)^2}{\sum_{i \in I_\beta^-} w_i (z_i^T \beta)^2} \leq \mu_w(\mathcal{D}) \leq \mu \\ \iff &\sum_{i \in I_\beta^+} w_i (z_i^T \beta)^2 \leq \mu \sum_{i \in I_\beta^-} w_i (z_i^T \beta)^2, \end{aligned}$$

which proves the second inequality.

Considering that the labeling of a dataset is arbitrary, i.e. we could always switch the 1 labels for the 0 labels and vice versa (if we flip the sign of β accordingly), the following relationship is true as well:

$$\begin{aligned} &\frac{\sum_{i \in I_\beta^-} w_i (z_i^T \beta)^2}{\sum_{i \in I_\beta^+} w_i (z_i^T \beta)^2} \leq \mu_w(\mathcal{D}) \leq \mu \\ \iff &\sum_{i \in I_\beta^-} w_i (z_i^T \beta)^2 \leq \mu \sum_{i \in I_\beta^+} w_i (z_i^T \beta)^2 \\ \iff &\mu^{-1} \sum_{i \in I_\beta^-} w_i (z_i^T \beta)^2 \leq \sum_{i \in I_\beta^+} w_i (z_i^T \beta)^2, \end{aligned}$$

which proves the first inequality. □

3.2 The Sensitivity Framework

After having imposed some reasonable restrictions on the datasets under study, it is now time to think about how an algorithm that selects a coreset $\mathcal{C} \subseteq \mathcal{D}$ could be constructed. One of the first ideas that come to mind to solve such a problem is the process of random sampling. After all, why don't we just randomly select a subset of points from $\mathcal{D}$ of the desired size? Wouldn't that already solve our problem?

The issue with this approach is that it cannot be guaranteed that such a uniform random sample will yield a coreset, i.e. a subset of $\mathcal{D}$ that helps us to obtain a $(1 \pm \epsilon)$-approximation of the original loss function. As we will later see in the experiments section, uniform sampling works reasonably well when the data is well behaved, but fails terribly when there are a few very important datapoints that it tends to miss. Intuitively speaking, if there are lots of points in a dataset that don't influence the loss function much, but only a few points that have a big impact on the loss function, uniform sampling fails because it tends to miss the few very important points.

It turns out, that one way to remedy the downsides of uniform sampling is to include a measure of importance in the sampling distribution. Instead of sampling each datapoint with equal probability, why don't we construct our sampling distribution in such a way, that the impact of each point on the loss function is taken into account? This way, more important points would be assigned a higher probability of being sampled and less important points would conversely be assigned a lower sampling probability. This

idea forms the basis of the so called *sensitivity framework*, an algorithmic framework introduced in [Feldman and Langberg, 2011] (see also [Feldman et al., 2020]), that aims to find coresets by randomly sampling points proportional to their importance for the loss function.

In the sensitivity framework, the importance of a point in a dataset can be thought of the maximum proportion of the loss function that it can take up in the worst case. To formalize this intuition, the sensitivity framework shifts the representation of a dataset as a collection of points towards the representation as a collection of *functions*, where each function represents the loss of a point. To explain what that means, consider the dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ with scaled model matrix Z, i.e. the rows of Z are given by the vectors $z_i = -(2y_i - 1)x_i$ and the loss function is given by $f(\beta) = \sum_{i=1}^n w_i g(z_i^T \beta)$, where $w_1, ..., w_n$ are positive weights. From now on, we will assign to each point in the dataset the function $g_i(\beta) = g(z_i^T \beta)$, that represents its individual contribution to the overall loss function. This way, we can equivalently represent the dataset $\mathcal{D}$ as the set of functions $F = \{g_1, ..., g_n\}$.

Having made this conceptual change of representing a dataset as a set of functions, we can now use this new representation to formalize the concept of importance for each point, according to wich we later want to sample. As already hinted at, the importance of a point will be the maximum share of the loss function, that the loss of the specific point will take up in the worst case. This worst case importance is also called the *sensitivity* of a point, and it was first introduced in [Langberg and Schulman, 2010]. A formal definition of this concept, which forms the basis of the sensitivity framework, is given below.

Definition 6 ([Langberg and Schulman, 2010])**.** *Let $F = \{g_1, ..., g_n\}$ be a set of functions, $g_i : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$, $i \in [n]$ and let $w \in \mathbb{R}^n_{>0}$ be a vector of positive weights. The sensitivity of g_i for $f_w(\beta) = \sum_{i=1}^n w_i g_i(\beta)$ is defined as*

$$\varsigma_i = \sup_{\beta \in \mathbb{R}^d,\ f_w(\beta) > 0} \frac{w_i g_i(\beta)}{f_w(\beta)}.$$

The total sensitivity, i.e. the sum of the sensitivities is $\mathfrak{S} = \sum_{i=1}^n \varsigma_i$.

The idea behind the sensitivity framework is to sample from a distribution where the sampling probabilities are determined by the sensitivities, i.e. the worst case importance of each point in the dataset. But there is one issue remaining: The true sensitivities $\varsigma_1, ..., \varsigma_n$ are unknown, and as pointed out in [Braverman et al., 2016], their computation requires solving the original optimization problem, which we wanted to avoid in the first place. Luckily, there exists a simple workaround: Instead of sampling proportionally to the true sensitivities, we can instead also sample proportionally to upper bounds s_i, where $s_i \geq \varsigma_i$, that are potentially easier to compute.

There is one caveat though, that we have to take into account: We will later see, that in the sensitivity framework, the size of the coreset that we obtain through sampling proportionally to upper bounds of the sensitivities, is directly influenced by the sum of

said bounds: $S = \sum_{i=1}^{n} s_i$. It follows, that we have to find bounds that are as tight as possible, so that our coreset won't be unnecessarily large.

The way in wich the authors of [Feldman and Langberg, 2011] were able to show that sampling proportionally to upper bounds of the sensitivities can lead to provably small coresets, was to relate the concept of sensitivities to the theory of so-called *range spaces* and the *VC-dimension* (see for example [Kearns and Vazirani, 1994] for an introduction of the VC-dimension). We introduce these concepts in the following definitions, because they will also turn out to be of crucial importance to the size of our coresets.

Definition 7 ([Feldman and Langberg, 2011])**.** *A range space is a pair* $\mathfrak{R} = (F, \text{ranges})$*, where* F *is a set and* ranges *is a family (set) of subsets of* F*.*

Definition 8 ([Feldman and Langberg, 2011])**.** *The VC-dimension* $\Delta(\mathfrak{R})$ *of a range space* $\mathfrak{R} = (F, \text{ranges})$ *is the size* $|G|$ *of the largest subset* $G \subseteq F$ *such that*

$$|\{G \cap R \mid R \in \text{ranges}\}| = 2^{|G|},$$

i.e. G *is shattered by* ranges*.*

The most important part to understand about the two preceding definitions is the concept of *shattering*. Given a set F, a set of subsets of F called ranges and a subset $G \subseteq F$, what does it mean if G is shattered by ranges? Definition 8 says, that the set of intersections between G and ranges must be equal to $2^{|G|}$, which is exactly the size of the set of all subsets of G. It follows, that for G to be shattered by ranges, every subset of G must appear in (be a subset of) at least one of the sets in ranges. From this it also immediately follows, that if G is shattered by ranges, then every subset of G is also shattered by ranges. Thus, the VC-dimension of the range space that is given by F and ranges is simply the size of the largest subset $G \subseteq F$, such that every subset of G appears in at least one element of ranges.

Instead of dealing with arbitrary sets like in definition 7 and definition 8, we are specifically dealing with a set of functions $F = \{g_1, ..., g_n\}$, that represents our dataset. In the next definition, we will see, how such a set of functions can be used to *induce* a range space, which will be of crucial importance when limiting the size of our coresets later on.

Definition 9 ([Feldman and Langberg, 2011])**.** *Let* F *be a finite set of functions mapping from* $\mathbb{R}^d$ *to* $\mathbb{R}_{\geq 0}$*. For every* $\beta \in \mathbb{R}^d$ *and* $r \geq 0$*, let*

$$\text{range}(F, \beta, r) = \{f \in F \mid f(\beta) \geq r\}$$

and let

$$\text{ranges}(F) = \left\{\text{range}(F, \beta, r) \mid \beta \in \mathbb{R}^d,\ r \geq 0\right\}.$$

Then we call $\mathfrak{R}_F := (F, \text{ranges}(F))$ *the range space induced by* F*.*

To understand what it means for a set of functions F to induce a range space, consider that we can always partition the set of functions into two disjoint subsets by choosing

a specific β and applying a threshold r: Every element in F with a value of $f(\beta) \geq r$ goes to one subset, the remainder goes to the other subset. Considering that our functions actually represent datapoints, each value of β and r give one way to partition the datapoints into two disjoint subsets. Thus, the set ranges(F) in definition 9 can be thought of the set of all possible partitions of functions (datapoints), that can be obtained for all possible $\beta \in \mathbb{R}^d$ and $r \geq 0$ and the VC-dimension of the induced range space is the size of the largest set of datapoints, that can be arbitrarily partitioned by different values of β and r.

We are now ready for the full theorem that relates sensitivity sampling to the theory of range spaces and the VC-dimension, which forms the core of the sensitivity framework. Its original version goes back to [Feldman and Langberg, 2011] and it was further improved in [Braverman et al., 2016]. The version we are presenting here is a slightly adapted variant introduced in [Feldman et al., 2020]:

Theorem 4 ([Feldman et al., 2020])**.** *Let $F = \{g_1, ..., g_n\}$ be a finite set of functions mapping from $\mathbb{R}^d$ to $\mathbb{R}_{\geq 0}$. Let $w \in \mathbb{R}^n_{>0}$ be a vector of positive weights. Let $\epsilon, \delta \in (0, \frac{1}{2})$. Let $s_i \geq \varsigma_i$ be upper bounds of the sensitivities and let $S = \sum_{i=1}^n s_i$. Given s_i, one can compute in time $O(|F|)$ a set $R \subseteq F$ of*

$$|R| \in O\left(\frac{S}{\epsilon^2}\left(\Delta \log S + \log\left(\frac{1}{\delta}\right)\right)\right)$$

weighted functions, such that with probability $1-\delta$ we have for all $\beta \in \mathbb{R}^d$ simultaneously

$$(1-\epsilon)\sum_{g_i \in F} w_i g_i(\beta) \leq \sum_{g_i \in R} u_i g_i(\beta) \leq (1+\epsilon)\sum_{g_i \in F} w_i g_i(\beta).$$

Each element of R is sampled independently with probability $p_j = \frac{s_j}{S}$ from F, $u_i = \frac{S w_j}{s_j |R|}$ denotes the weight of a function $g_i \in R$ that corresponds to $g_j \in F$ and Δ is an upper bound on the VC-dimension of the range space $\mathfrak{R}_{F^*}$ induced by F^*, where F^* is the set of functions $g_i \in F$ scaled by $\frac{S w_i}{s_i |R|}$, i.e. $F^* = \left\{\frac{S w_i}{s_i |R|} g_i(\beta) \mid i \in [n]\right\}$.

The set of functions $R \subseteq F$ with weights u is of course a representation of the coreset that we are interested in. As theorem 4 tells us, the size of this coreset depends on both, the sum of the sensitivity bounds as well as the VC-dimension of the range space induced by F^*, a reweighted version of F.

Another thing to note is that theorem 4 introduces a failure probability δ, which also influences the size of the coreset. The reason why we need this new parameter becomes clear when considering, that the coreset is selected by random sampling, i.e. the coreset-construction process is probabilistic and probabilistic processes can fail. This failure probability is reflected by the parameter δ.

Equipped with theorem 4, we now have a clear roadmap to follow in the pursuit of our goal of finding a coreset construction algorithm. We know, that we have to find small upper bounds on the sensitivities of our function set $F = \{g_1, ..., g_n\}$ for the purpose of random sampling and at the same time, we also know that we have to control the VC-dimension of the range space of F^*. Thus, bounding the sensitivities as well as bounding the VC-dimension are the main challenges of the next section.

3.3 Constructing the Coreset

Without any point of reference, the task of finding tight and efficiently computable upper bounds on the sensitivities seems rather challenging. How are we supposed to find those ominous bounds and on top of that, how can we make sure that their sum will be small? It helps to remind ourselves of the original problem that the sensitivities were designed to solve. Instead of sampling every point with equal probability, the sensitvities were introduced to include the importance of each point into the sampling process. But what if similar importance distributions already existed? Could it be possible to choose an existing importance distribution and relate it to the concept of sensitivities in order to obtain upper bounds?

It turns out, that there is one importance sampling distribution that is particularly helpful in the context of our problem: The so-called statistical leverage scores (see for example [Drineas et al., 2012]). For a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ with model matrix X, the statistical leverage score of the i-th observation is given by $\ell_i = x_i^T (X^T X)^{-1} x_i$. Intuitively speaking, ℓ_i is a measure of the uniqueness of a given observation x_i. If there aren't many data points close to x_i, i.e. we can consider x_i to be unique, then the leverage score of x_i is high. On the other hand, if there are a lot of other data points close to x_i, i.e. x_i is not unique, then the leverage score is low. When using leverage scores as a sampling distribution, we give a higher weight to points that are unique and different from the other points, compared to points that are surrounded by a lot of similar other points.

To better understand what that means, a visualization of the statistical leverage scores of an artificially generated two dimensional dataset is given in figure 2 as an example. We can see, that most of the datapoints that are crowding the center of the dataset have rather low leverage scores, but the further out the points are located relatively to the center, the higher the leverage scores become. Particularly, there is a small group of outliers in the top right of the coordinate system, and we can see that their leverage scores are almost four times as high as the leverage scores of the points in the center. Thus, when sampling points proportionally to their leverage scores, we are less likely to miss outliers in the data that could potentially have a high impact on the loss function.

An alternative way of defining the leverage scores, which will be particularly important to us when mathematically deriving the sensitivity bounds later on, is to specify them as the squared row norms of an orthonormal basis of the model matrix $X \in \mathbb{R}^{n \times d}$. Such an orthonormal basis can for example be obtained by computing a so-called QR-decomposition of X (see for example [Golub and van Loan, 2013]), where $X = QR$ is factorized into an orthonormal matrix $Q \in \mathbb{R}^{n \times d}$ and an upper triangular matrix $R \in \mathbb{R}^{d \times d}$.

Our goal in this section is to adapt the leverage scores in such a way, that we can obtain tight upper bounds on the sensitivities. But we are not quite ready for that yet. Before we can get there, we first have to focus our attention back to the probit loss function $g(x)$, because it turns out that in order to bound the sensitivities by using the leverage scores, we first have to cover some important properties of $g(x)$.

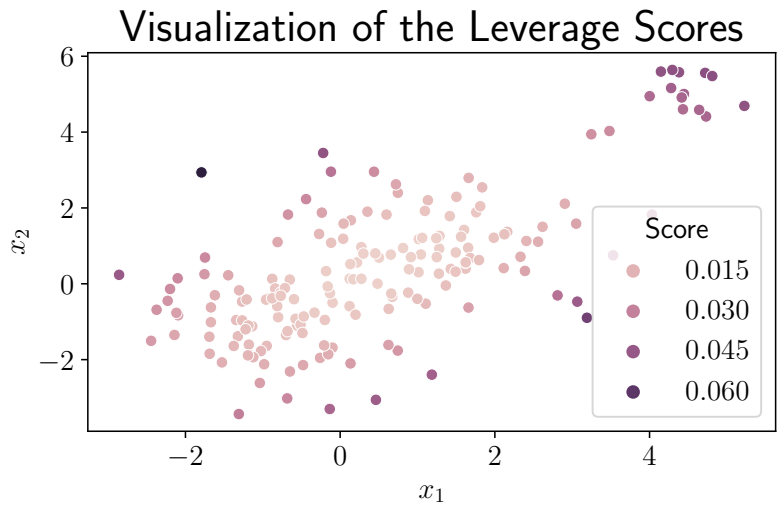

Figure 2: Visualization of the statistical leverage scores of an artificial two dimensional dataset. Darker colors indicate a higher leverage score than lighter colors.

3.3.1 A closer examination of the probit loss

In order to find bounds on the sensitivities, we also need bounds on the probit loss $g(x) = \ln\left(\frac{1}{1-\Phi(x)}\right)$. The first bound that we will derive holds for all $x \geq 0$ and shows that $g(x)$ grows at least like a quadratic function. Next, we will show that for all $x \geq 2$, $g(x)$ is upper bounded by a quadratic function, and thus $g(x)$ asymptotically grows like a quadratic function. Both of these bounds will turn out to be helpful later on. They are proven in lemma 2 and lemma 3.

Lemma 2. *Let $g(x) = \ln\left(\frac{1}{1-\Phi(x)}\right)$. Then, for all $x \geq 0$, it holds that:*

$$\frac{1}{2}x^2 \leq g(x).$$

Proof. We first show the claim for all $x \geq 1$, by using the following inequality:

$$\begin{aligned}
\Phi(-x) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{-x} \exp\left(-\frac{1}{2}t^2\right) dt \\
&\leq \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{-x} -t \exp\left(-\frac{1}{2}t^2\right) dt \\
&= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right) \\
&\leq \exp\left(-\frac{1}{2}x^2\right).
\end{aligned}$$

In the next step, we use this inequality to show that for $x \geq 1$:

$$e^{g(x)} = e^{\ln\left(\frac{1}{1-\Phi(x)}\right)} = \frac{1}{\Phi(-x)} \geq e^{\frac{1}{2}x^2}$$
$$\iff$$
$$g(x) \geq \frac{1}{2}x^2,$$

which proves the bound for $x \geq 1$.

Let us now turn to the case when $0 \leq x \leq 1$. Both $g(x)$ and $\frac{1}{2}x^2$ are monotonically increasing and continuous functions for $0 \leq x \leq 1$. Making use of the fact that $g(0) > \frac{1}{2}$, it follows for all $0 \leq x \leq 1$, that

$$g(x) \geq g(0) > \frac{1}{2} = \max_{0 \leq x \leq 1} \frac{1}{2}x^2 \geq \frac{1}{2}x^2,$$

which concludes the proof. □

Lemma 3. *Let $g(x) = \ln\left(\frac{1}{1-\Phi(x)}\right)$. Then, for all $x \geq 2$, it holds that:*

$$g(x) \leq x^2.$$

Proof. In [Gordon, 1941], it was shown that the following inequality holds for all $x \geq 0$:

$$\Phi(-x) \geq \frac{1}{\sqrt{2\pi}} \frac{x}{x^2+1} e^{-\frac{1}{2}x^2}.$$

We can use this inequality to establish that for all $x \geq 2$ it holds that:

$$\begin{aligned}
e^{x^2} \cdot \Phi(-x) &\geq e^{x^2} \frac{1}{\sqrt{2\pi}} \frac{x}{x^2+1} e^{-\frac{1}{2}x^2} \\
&= e^{\frac{1}{2}x^2} \frac{1}{\sqrt{2\pi}} \frac{x}{x^2+1} \\
&= e^{\frac{1}{2}x^2} \frac{1}{\frac{4}{3}(x^2+1)} \frac{\frac{4}{3}x}{\sqrt{2\pi}} \\
&\geq \frac{e^{\frac{1}{2}x^2}}{\frac{4}{3}(x^2+1)} \\
&\geq \frac{e^{\frac{1}{2}x^2}}{e^{\frac{1}{2}x^2}} \\
&= 1 \\
\iff & \\
e^{x^2} &\geq \frac{1}{1-\Phi(x)} \\
\iff & \\
x^2 &\geq \ln\left(\frac{1}{1-\Phi(x)}\right) = g(x),
\end{aligned}$$

which completes the proof. □

3.3.2 Finding the sensitivity bounds

Having successfully established the quadratic bounds on the probit loss $g(x)$, we can now turn our attention back to the task of finding upper bounds on the sensitivities. As already mentioned, we will be using the statistical leverage scores in order to do so.

To this end, we will show that the function set $F = \{g_1, ..., g_n\}$, which represents our dataset, can be partitioned into two classes of functions for which we can find upper bounds on the sensitivities. The first class will contain all points, where the loss funcion surpasses a specific threshold. It turns out, that for a given β, the threshold $z_i^T\beta \geq 2$ is a suitable candidate. In the next lemma, we show how we can relate the loss function of all the points in this class to the statistical leverage scores, while also incorporating μ into the upper bound. To do this, we use the notion of leverage scores as the squared row norms of an orthogonal basis, that can be obtained for example by conducting a QR-factorization.

Lemma 4. *Let $\mathcal{D}$ be a d-dimensional and μ-complex dataset of size $|\mathcal{D}| = n$ with scaled model matrix $Z \in \mathbb{R}^{n\times d}$ and let $w \in \mathbb{R}^n_{>0}$ be a vector of positive weights. Let $F = \{g_1, ..., g_n\}$ be a set of functions with $g_i(\beta) = g(z_i^T\beta)$ and let $f_w(\beta) = \sum_{i=1}^n w_i g_i(\beta)$. Then, it holds that*

$$w_j g_j(\beta) \leq 2\|U_j\|_2^2(1+\mu)f_w(\beta) \quad \forall\ j \in \{i \in [n] :\ z_i^T\beta \geq 2\},$$

where $U \in \mathbb{R}^{n\times d}$ is an orthonormal basis for the columnspace of $\sqrt{D_w}Z$ and $\sqrt{D_w} \in \mathbb{R}^{n\times n}$ is a diagonal matrix, where the i-th diagonal element is equal to $\sqrt{w_i}$ and $U_j \in \mathbb{R}^d$ is the j-th row of U.

Proof. Let $\sqrt{D_w}Z = UR$, where U is an orthonormal basis for the columnspace of $\sqrt{D_w}Z$. Then, for all $j \in \{i \in [n] :\ z_i^T\beta \geq 2\}$:

$$w_j g_j(\beta) = w_j g(z_j^T\beta) = w_j g\left(\frac{\sqrt{w_j}z_j^T\beta}{\sqrt{w_j}}\right) = w_j g\left(\frac{U_j^T R\beta}{\sqrt{w_j}}\right) \leq w_j g\left(\frac{\|U_j\|_2\|R\beta\|_2}{\sqrt{w_j}}\right),$$

where $U_j \in \mathbb{R}^d$ is the vector that constitutes the j'th row of U and the inequality is true due to the *Cauchy-Schwarz inequality*. We continue the proof as follows:

$$\begin{aligned} w_j g\left(\frac{\|U_j\|_2\|R\beta\|_2}{\sqrt{w_j}}\right) &= w_j g\left(\frac{\|U_j\|_2\|UR\beta\|_2}{\sqrt{w_j}}\right) \\ &= w_j g\left(\frac{\|U_j\|_2\|\sqrt{D_w}Z\beta\|_2}{\sqrt{w_j}}\right) \\ &\leq \|U_j\|_2^2\|\sqrt{D_w}Z\beta\|_2^2 \\ &= \|U_j\|_2^2\sum_{i=1}^n w_i(z_i^T\beta)^2. \end{aligned}$$

Here, the first equality follows from the fact that U is orthonormal, i.e. multiplying by U doesn't change the norm of a vector. The inequality follows from the bound $g(x) \leq x^2$ that holds for all $x \geq 2$, which was shown in lemma 3.

Now, let $I_\beta^+ = \{i \in [n] : \ w_i z_i^T \beta > 0\}$ and let $I_\beta^- = \{i \in [n] : \ w_i z_i^T \beta < 0\}$ like in definition 5, the definition of μ-complexity. We continue the proof by making use of the relationship that was shown in lemma 1:

$$\begin{aligned}
\|U_j\|_2^2 \sum_{i=1}^n w_i (z_i^T \beta)^2 &= \|U_j\|_2^2 \left(\sum_{i \in I_\beta^+} w_i (z_i^T \beta)^2 + \sum_{i \in I_\beta^-} w_i (z_i^T \beta)^2 \right) \\
&\leq \|U_j\|_2^2 \left(\sum_{i \in I_\beta^+} w_i (z_i^T \beta)^2 + \mu \sum_{i \in I_\beta^+} w_i (z_i^T \beta)^2 \right) \\
&= \|U_j\|_2^2 (1+\mu) \sum_{i \in I_\beta^+} w_i (z_i^T \beta)^2 \\
&\leq 2\|U_j\|_2^2 (1+\mu) \sum_{i \in I_\beta^+} w_i g(z_i^T \beta),
\end{aligned}$$

where the last inequality follows from the bound $g(x) \geq \frac{1}{2}x^2$, that holds for all $x \geq 0$, which we proved in lemma 2.

From here, we can use the fact that g is a strictly positive function to complete the proof:

$$2\|U_j\|_2^2 (1+\mu) \sum_{i \in I_\beta^+} w_i g(z_i^T \beta) \leq 2\|U_j\|_2^2 (1+\mu) \sum_{i=1}^n w_i g(z_i^T \beta) = 2\|U_j\|_2^2 (1+\mu) f_w(\beta)$$

□

In the next lemma, we turn to the remaining class of points where $z_i^T \beta \leq 2$, and show how the sensitivity of these points can be bounded by a constant value that only depends on μ and the weight of the given point.

Lemma 5. *Let $\mathcal{D}$ be a d-dimensional and μ-complex dataset of size $|\mathcal{D}| = n$ with scaled model matrix $Z \in \mathbb{R}^{n \times d}$ and let $w \in \mathbb{R}^n_{>0}$ be a vector of positive weights. Let $F = \{g_1, ..., g_n\}$ be a set of functions with $g_i(\beta) = g(z_i^T \beta)$ and let $f_w(\beta) = \sum_{i=1}^n w_i g_i(\beta)$. Then, it holds that*

$$w_j g_j(\beta) \leq \frac{w_j}{\mathcal{W}} (80 + 16\mu) f_w(\beta) \quad \forall \ j \in \{i \in [n] : \ z_i^T \beta \leq 2\},$$

where $\mathcal{W} = \sum_{i=1}^n w_i$ is the sum of all weights.

Proof. We first start by noting that $g(-1) > \frac{1}{10}$ and that $g(2) < 4$. Now, we partition the indices into two sets as follows:

$$\begin{aligned}
K_\beta^- &= \{i \in [n] \mid z_i^T \beta \leq -1\} \\
K_\beta^+ &= \{i \in [n] \mid z_i^T \beta > -1\}.
\end{aligned}$$

In the case that $\sum_{j \in K_\beta^+} w_j \geq \frac{1}{2}\mathcal{W}$, the following relationship holds:

$$f_w(\beta) = \sum_{i=1}^{n} w_i g(z_i^T \beta) \geq \sum_{i \in K_\beta^+} w_i g(z_i^T \beta) \geq \frac{\sum_{i \in K_\beta^+} w_i}{10} \geq \frac{\mathcal{W}}{20} = \frac{\mathcal{W}}{20 w_j} w_j \geq \frac{\mathcal{W}}{80 w_j} w_j g(z_j^T \beta),$$

where $j \in \{i \in [n] : \ z_i^T \beta \leq 2\}$. Thus, we have in this case:

$$w_j g(z_j^T \beta) \leq \frac{80 w_j}{\mathcal{W}} f_w(\beta).$$

If on the other hand $\sum_{j \in K_\beta^-} w_j \geq \frac{1}{2}\mathcal{W}$, we have that

$$f_w(\beta) = \sum_{i=1}^{n} w_i g(z_i^T \beta) \geq \sum_{i \in I_\beta^+} w_i g(z_i^T \beta) \geq \frac{1}{2} \sum_{i \in I_\beta^+} w_i (z_i^T \beta)^2 \geq \frac{1}{2\mu} \sum_{i \in I_\beta^-} w_i (z_i^T \beta)^2,$$

where $I_\beta^+ = \{i \in [n] : \ w_i z_i^T \beta > 0\}$ and $I_\beta^- = \{i \in [n] : \ w_i z_i^T \beta < 0\}$ like in definition 5 (μ-complexity). The second inequality is true due to the lower bound $g(x) \geq \frac{1}{2}x^2$ that holds for all $x \geq 0$ (see lemma 2) and the third inequality is true due to a property of μ that was proved in lemma 1.

We continue the proof as follows:

$$\frac{1}{2\mu} \sum_{i \in I_\beta^-} w_i (z_i^T \beta)^2 \geq \frac{1}{2\mu} \sum_{i \in K_\beta^-} w_i (z_i^T \beta)^2 \geq \frac{1}{2\mu} \sum_{i \in K_\beta^-} w_i \geq \frac{\mathcal{W}}{4\mu} \geq \frac{\mathcal{W}}{16 \mu w_j} w_j g(z_j^T \beta),$$

which leads us to the upper bound for the second case:

$$w_j g(z_j^T \beta) \leq \frac{16 \mu w_j}{\mathcal{W}} f_w(\beta).$$

We can conclude the proof by adding both upper bounds:

$$w_j g_j(\beta) = w_j g(z_j^T \beta) \leq \frac{80 w_j}{\mathcal{W}} f_w(\beta) + \frac{16 \mu w_j}{\mathcal{W}} f_w(\beta) = \frac{w_j}{\mathcal{W}} (80 + 16\mu) f_w(\beta).$$

□

It is now time to use the results from lemma 4 and lemma 5 to derive an upper bound on the sensitivities. Because we showed how the dataset can be partitioned in "high loss points" and "low loss points" for every given β and how these two classes of points can both be upper bounded, it simply suffices to add those bounds together to bound the sensitivities for any given β. As a final step, we only have to show that the total sum of the sensitivities remains small. We do both in the following lemma.

Lemma 6. *Let $\mathcal{D}$ be a d-dimensional and μ-complex dataset of size $|\mathcal{D}| = n$ with scaled model matrix $Z \in \mathbb{R}^{n \times d}$, let $w \in \mathbb{R}^n_{>0}$ be a vector of positive weights and let $U \in \mathbb{R}^{n \times d}$ be an orthonormal basis for the columnspace of $\sqrt{D_w} Z$. Let $F = \{g_1, ..., g_n\}$ be a set of*

functions with $g_i(\beta) = g(z_i^T\beta)$ *and let* $f_w(\beta) = \sum_{i=1}^n w_i g_i(\beta)$*. Then, the sensitivity* ς_i *of* g_i *(see definition 6) is upper bounded by*

$$\varsigma_i \leq s_i = (80 + 16\mu)(\|U_i\|_2^2 + \frac{w_i}{\mathcal{W}}),$$

and the total sensitivity is bounded by

$$\mathfrak{S} = \sum_{i=1}^n \varsigma_i \leq 192\mu d.$$

Proof. We can use the bounds that we derived in lemma 4 and lemma 5 to bound the sensitivities:

$$\begin{aligned}
\varsigma_i &= \sup_{\beta\in\mathbb{R}^d,\ f_w(\beta)>0} \frac{w_i g(z_i\beta)}{f_w(\beta)} \\
&\leq \sup_{\beta\in\mathbb{R}^d,\ f_w(\beta)>0} \frac{2\|U_i\|_2^2(1+\mu)f_w(\beta) + \frac{w_i}{\mathcal{W}}(80+16\mu)f_w(\beta)}{f_w(\beta)} \\
&= 2\|U_i\|_2^2(1+\mu) + \frac{w_i}{\mathcal{W}}(80+16\mu) \\
&\leq \|U_i\|_2^2(80+16\mu) + \frac{w_i}{\mathcal{W}}(80+16\mu) \\
&= (80+16\mu)(\|U_i\|_2^2 + \frac{w_i}{\mathcal{W}}),
\end{aligned}$$

which completes the first part of the proof. For the next part, we use that U is an orthonormal matrix. The Frobenius norm $\|U\|_F$ (see for example [Golub and van Loan, 2013]) of an orthonormal matrix is equal to $\sqrt{d}$, as can easily be verified:

$$\|U\|_F = \sqrt{\sum_{k=1}^d \sum_{l=1}^n |u_{lk}|^2} = \sqrt{\sum_{k=1}^d 1} = \sqrt{d},$$

where the second equality follows from the fact that the columns of U have unit norm due to its orthonormality. We can now conclude the proof as follows:

$$\begin{aligned}
\mathfrak{S} = \sum_{i=1}^n \varsigma_i &\leq (80+16\mu)\sum_{i=1}^n \|U_i\|_2^2 + \frac{w_i}{\mathcal{W}} \\
&= (80+16\mu)(\|U\|_F^2 + 1) \\
&= (80+16\mu)(d+1) \\
&\leq 96\mu(d+1) \\
&\leq 192\mu d.
\end{aligned}$$

□

We now have successfully completed the first task on our list of developing a coreset construction algorithm by using the sensitivity framework. Not only did we manage to derive upper bounds on the sensitivities of the function set $F = \{g_1, ..., g_n\}$ that represents our dataset by using the statistical leverage scores, but we also showed that the sum of those bounds is in $O(\mu d)$, even independent of the total number of datapoints n. The final step, before putting everything together, is now to find an upper bound of the VC-dimension of the range-space induced by F^*.

3.3.3 Bounding the VC-dimension

Remember, that in the core theorem of the sensitivity framework (see theorem 4), the function-set that induces our range space of interest is defined as

$$F^* = \left\{ \frac{Sw_i}{s_i|R|} g_i(\beta) \mid i \in [n] \right\},$$

where s_i are upper bounds on the sensitivities, S is the sum of these upper bounds, $|R|$ is the size of the sample and w_i are the initial weights. We are thus dealing with a set of weighted probit loss functions.

In order to approach the complex problem of bounding the VC-dimension of a set of arbitrarily weighted probit loss functions, we first deal with the slightly simpler problem that arises when the weights only consist of a single positive constant, i.e. we are looking at the set $\mathcal{F}^c = \{cg_i(\beta) \mid i \in [n]\}$. The authors of [Huggins et al., 2016] showed, that in the case of the logistic loss, it is possible to relate the VC-dimension of the range space induced by $\mathcal{F}^c$ to the VC-dimension of the affine hyperplane classifier, which the authors of [Kearns and Vazirani, 1994] showed to be bounded by $d+1$. It turns out, that a similar case can be made for the probit loss, which we demonstrate in the following lemma.

Lemma 7 (cf. [Huggins et al., 2016])**.** *Let $Z \in \mathbb{R}^{n \times d}$, let $z_i \in \mathbb{R}^d$ be the i-th row of Z and let $c \in \mathbb{R}_{>0}$. Let $F = \{g_1, ..., g_n\}$ be a set of functions with $g_i(\beta) = g(z_i^T \beta)$, where $g(x) = \ln\left(\frac{1}{1-\Phi(x)}\right)$ is the probit loss. The VC-dimension of the range space induced by*

$$\mathcal{F}^c = \{cg_i(\beta) \mid i \in [n]\}$$

is bounded by $\Delta(\mathfrak{R}_{\mathcal{F}^c}) \leq d + 1$.

Proof. We start by noting that for all $G \subseteq \mathcal{F}^c$ we have

$$|\{G \cap R \mid R \in \text{ranges}(\mathcal{F}^c)\}| = \left|\left\{\text{range}(G, \beta, r) \mid \beta \in \mathbb{R}^d,\ r \geq 0\right\}\right|.$$

Since g is invertible and monotone, we have for all $\beta \in \mathbb{R}^d$ and $r \geq 0$ that

$$\begin{aligned}
\text{range}(G, \beta, r) &= \{g_i \in G \mid g_i(\beta) \geq r\} \\
&= \left\{g_i \in G \mid cg(z_i^T \beta) \geq r\right\} \\
&= \left\{g_i \in G \mid z_i^T \beta \geq g^{-1}\left(\frac{r}{c}\right)\right\}.
\end{aligned}$$

Note, that $\left\{g_i \in G \mid z_i^T \beta \geq g^{-1}\left(\frac{r}{c}\right)\right\}$ corresponds to the positively classified points of the affine hyperplane classifier $x \mapsto \text{sign}\left(x^T\beta - g^{-1}\left(\frac{r}{c}\right)\right)$. We thus have for all $G \subseteq \mathcal{F}^c$, that

$$|\{G \cap R \mid R \in \text{ranges}(\mathcal{F}^c)\}| = \left|\left\{\left\{g_i \in G \mid z_i^T\beta - s \geq 0\right\} \mid \beta \in \mathbb{R}^d,\ s \in \mathbb{R}\right\}\right|.$$

As shown in [Kearns and Vazirani, 1994], the VC-dimension of the set of affine hyperplane classifiers is $d+1$, so it follows that $\Delta(\mathfrak{R}_{\mathcal{F}^c}) \leq d+1$, which concludes the proof. □

In the next step, we will generalize the class $\mathcal{F}^c$ of constantly weighted probit loss functions to the class $\mathcal{F}^w = \{w_i g_i(\beta) \mid i \in [n]\}$ of arbitrarily weighted probit loss functions for a weight vector $w \in \mathbb{R}^n_{>0}$, which also includes F^*, our class of interest. The authors of [Munteanu et al., 2018] presented a proof that shows how the VC-dimension of the range space induced by such a set can be bounded by $t \cdot (d+1)$ in the case of logistic regression, where $t \in \mathbb{N}$ is the number of distinct weights in the vector w. We follow a similar path and adapt their argument to the context of probit regression in the following lemma.

Lemma 8 (cf. [Munteanu et al., 2018])**.** *Let $Z \in \mathbb{R}^{n\times d}$, let $z_i \in \mathbb{R}^d$ be the i-th row of Z and let $w \in \mathbb{R}^n_{>0}$ be a vector of positive weights, where $w_i \in \{v_1, ..., v_t\}$ for all $i \in [n]$. Let $F = \{g_1, ..., g_n\}$ be a set of functions with $g_i(\beta) = g(z_i^T\beta)$. The VC-dimension of the range space induced by*

$$\mathcal{F}^w = \{w_i g_i(\beta) \mid i \in [n]\}$$

is bounded by $\Delta(\mathfrak{R}_{\mathcal{F}^w}) \leq t \cdot (d+1)$.

Proof. We start by partitioning the functions in $\mathcal{F}^w$ into t disjoint classes

$$F_j = \{w_i g(z_i\beta) \in \mathcal{F}^w \mid w_i = v_j\}, \quad j \in [t].$$

The functions in each of these classes have an equal weight, wich means that by lemma 7, each of their induced range spaces has a VC-dimension of at most $d+1$.

For the sake of contradiction, assume that $\Delta(\mathfrak{R}_{\mathcal{F}^w}) > t \cdot (d+1)$ and let G be the corresponding set of size $|G| > t \cdot (d+1)$ that is shattered by ranges$(\mathcal{F}^w)$. Since the sets F_j are disjoint, each intersection $F_j \cap G$ must be shattered by ranges(F_j) as well. Further, at least one of the intersections must have at minimum $\frac{|G|}{t}$ elements, which means that for at least one $j \in [t]$ it holds that $|F_j \cap G| \geq \frac{|G|}{t} > \frac{t\cdot(d+1)}{t} = d+1$. This is a contradiction to lemma 7, which concludes the proof. □

We have now found a way to bound the VC-dimension of the range space induced by F^* in the number of distinct weights, i.e. the number of distinct values of $\frac{Sw_i}{s_i|R|}$. But there is one important issue that remains to be dealt with: In the general case, we don't know the number of distinct values of $\frac{Sw_i}{s_i|R|}$, and it is even reasonable to assume, that this value can be equal to the total number of datapoints, n. This would be a problem for us though, because the core theorem of the sensitivity framework (theorem 4) tells

us, that the size of our coreset will depend linearly on the VC-dimension of the range space induced by F^*. If this VC-dimension is in $O(n)$, our coresets won't be small. It follows, that we have to find a way to work around that problem in order to obtain small coresets.

3.3.4 A first naïve algorithm

It now remains to solve the final challenge: How can we limit the number of distinct weights in F^* in order to limit the VC-dimension of the range space induced by F^* and obtain small coresets? The authors of [Munteanu et al., 2018] were facing a similar situation, and it turns out that they managed to come up with a very clever idea: Their approach to the problem is to slightly increase the upper bounds on the sensitivities s_i to a new value $s'_i \geq s_i$, such that the fraction $\frac{s'_i}{w_i}$ is exactly a power of two. This way, the total sum of the sensitivities $S' = \sum_{i=1}^{n} s'_i$ is still under control, because as a worst case bound we have that $S' \leq 2S$, which doesnt influence the big-o notation and we still have $S' \in O(\mu d)$. The big advantage of this approach is, that we now have a way to bound the number of distinct values of $\frac{S'w_i}{s'_i|R|}$ in a term that is logarithmic in n. To see why this is the case, we first derive two simple inequalities. The first one goes like this and holds for all $i \in [n]$:

$$\frac{s'_i}{w_i} \leq \frac{2s_i}{w_i} = \frac{2(80+16\mu)(\|U_i\|_2^2 + \frac{w_i}{\mathcal{W}})}{w_i} \leq \frac{192\mu(\|U_i\|_2^2 + \frac{w_i}{\mathcal{W}})}{w_i} \leq \frac{384\mu}{w_i} \leq \frac{384\mu}{w_{min}},$$

where w_{min} is the minimum weight and $\mathcal{W}$ is the sum of all weights. The third inequality holds, because it is always true that $\mu \geq 1$ and the fourth inequality is true because one property of the statistical leverage scores is that $\|U_i\|_2^2 \leq 1$.

Next, we show how $\frac{s'_i}{w_i}$ can be lower-bounded for all $i \in [n]$:

$$\frac{s'_i}{w_i} \geq \frac{s_i}{w_i} \geq \sup_{\beta \in \mathbb{R}^d} \frac{g_i(\beta)}{\sum_{i=1}^{n} w_i g_i(\beta)} \overset{\beta=0}{\geq} \frac{1}{\sum_{i=1}^{n} w_i} \geq \frac{1}{n w_{max}},$$

where w_{max} is the maximum weight. Putting both of the inequalities together, we thus have that

$$\frac{1}{n w_{max}} \leq \frac{s'_i}{w_i} \leq \frac{384\mu}{w_{min}}.$$

We know, that the values of $\frac{s'_i}{w_i}$ are exactly powers of two, so we can compute the amount of possible distinct values of $\frac{s'_i}{w_i}$, which we call t, like this:

$$t \leq \log_2\left(\frac{384\mu}{w_{min}}\right) - \log_2\left(\frac{1}{n w_{max}}\right) = \log_2\left(384\mu n \frac{w_{max}}{w_{min}}\right) \in O\left(\log_2(\mu n \omega)\right),$$

where $\omega = \frac{w_{max}}{w_{min}}$. Thus, when sampling according to s'_i, our function class of interest becomes

$$F^{*\prime} = \left\{ \frac{S'w_i}{s'_i|R|} g_i(\beta) \mid i \in [n] \right\},$$

and the weights $\frac{S'w_i}{s'_i|R|}$ can assume only $O(\log_2(\mu n\omega))$ distinct values. Plugging this into lemma 8, we get that the VC-dimension of the range space induced by $F^{*'}$ is upper bounded by a term in $O(d\log_2(\mu n\omega))$.

We now have everything we need to construct our first algorithm. As we already hinted at, the algorithm will sample the points from our dataset proportionally to the rounded values values s'_i, i.e. each point is assigned a sampling probability

$$p_i = \frac{s'_i}{S'_i} = \frac{\left\lceil \ell_i + \frac{w_i}{\mathcal{W}} \right\rceil_2}{\sum_{i=1}^n \left\lceil \ell_i + \frac{w_i}{\mathcal{W}} \right\rceil_2},$$

where $\lceil . \rceil_2$ indicates the rounding of s_i such that $\frac{s'_i}{w_i}$ is a power of two and ℓ_i is the statistical leverage score of the i-th datapoint, which can be obtained through a QR decomposition of the row-wise weighted matrix $\sqrt{D_w}Z$. The resulting naïve algorithm is given in Algorithm 2.

Algorithm 2: Naïve coreset construction algorithm

Input: Dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ with scaled model matrix $Z \in \mathbb{R}^{n\times d}$ and weight vector $w \in \mathbb{R}^n_{>0}$, $\mathcal{W} = \sum_{i=1}^n w_i$, size parameter $k \in \mathbb{N}$

Output: A subset $\mathcal{C} \subseteq \mathcal{D}$ of size $|\mathcal{C}| = k$ with weight vector $u \in \mathbb{R}^k_{>0}$

Compute the QR-decomposition of $\sqrt{D_w}Z = QR$

for $i = 1, ..., n$ **do**

  $\ell_i = \|Q_i\|_2^2$, where Q_i is the i-th row of Q

  $a_i = \ell_i + \frac{w_i}{\mathcal{W}}$

  $s'_i = w_i 2^{\left\lceil \log_2\left(\frac{a_i}{w_i}\right)\right\rceil}$

for $i = 1, ..., n$ **do**

  $p_i = \frac{s'_i}{\sum_{i=1}^n s'_i}$

$\mathcal{C}^{(0)} = \emptyset$

for $i = 1, ..., k$ **do**

  Randomly sample a point c_j from $\mathcal{D}$ with probabilities $p_1, ..., p_n$

  $\mathcal{C}^{(i)} = \mathcal{C}^{(i-1)} \cup \{c_j\}$

  $u_i = \frac{w_j}{k \cdot p_j}$

return $\mathcal{C}^{(k)}$, $(u_1, ..., u_k)^T$

Although the naïve algorithm is a first proof of concept of how small coresets for probit regression can in principle be constructed, there are still multiple issues with this algorithm that we have to deal with. The first and most obvious issue is the QR decomposition of $\sqrt{D_w}Z$ that is needed in order to obtain the statistical leverage scores. When the dataset is small and fits into the main memory, the QR decomposition is not an issue, but as soon as the size of the data grows, its computation on the whole dataset becomes infeasible. Ideally, we would like to compute (or at least approximate) the QR decomposition in a single pass over the data, i.e. we only want look at each element of $\mathcal{D}$

once. Standard algorithms for computing a QR factorization (see for example the Givens method described in [Golub and van Loan, 2013]) don't scale well for large datasets, and usually require more than one pass over the data. Thus, one of the remaining challenges is to find a way to adapt the QR decomposition for large datasets in order to efficiently compute the leverage scores.

The second issue of the naïve algorithm lies in the random sampling procedure. Ideally, we would also like to perform the sampling according to the distribution defined by $p_1, ..., p_n$ in a single pass over the dataset. In the next section, we will further explore how to solve both of these issues and obtain two algorithms that only require two passes or one pass over the data, respectively. But before we get into that, we conclude this section with a proof, that the naïve algorithm is indeed a correct algorithm for constructing small coresets for probit regression.

Theorem 5. *Algorithm 2 returns a $(1 \pm \epsilon)$-coreset for probit regression, if the size parameter k satisfies*

$$k \in O\left(\frac{\mu d^2}{\epsilon^2} \log(\mu\omega n) \log(\mu d)\right).$$

Proof. We use the core theorem of the sensitivity framework (see theorem 4) in order to prove the claim.

In lemma 6, we showed that $s_1, ..., s_n$ are upper bounds on the sensitivities and that $S = \sum_{i=1}^{n} s_i \in O(\mu d)$. We also saw, that when rounding the s_i up in such a way that $\frac{s'_i}{w_i}$ is a power of two, $\sum_{i=1}^{n} s'_i \leq 2S \in O(\mu d)$. Further, we showed that the VC-dimension of the range space induced by $F^{*'}$ can be upper bounded by a term $\Delta \in O(d \log(\mu\omega n))$, where $\omega = \frac{w_{max}}{w_{min}}$ is the ratio of the largest and smallest weight. Setting the failure probability equal to $\delta = n^{-c}$ for any absolute constant $c > 1$, we can plug everything into the core theorem of the sensitivity framework and obtain the desired coreset size:

$$\begin{aligned} k &\in O\left(\frac{S'}{\epsilon^2}\left(\Delta \log S' + \log\left(\frac{1}{\delta}\right)\right)\right) \\ &\subseteq O\left(\frac{\mu d}{\epsilon^2}\left(d \log(\mu\omega n) \log(\mu d) + \log(n^c)\right)\right) \\ &\subseteq O\left(\frac{\mu d^2}{\epsilon^2} \log(\mu\omega n) \log(\mu d)\right) \end{aligned}$$

□

4 Efficient Coreset Algorithms

When constructing the naïve algorithm, we encountered two main challenges that have to be dealt with in order to make the algorithm more efficient and suitable for large datasets: First, we have to find a way to efficiently compute the leverage scores, preferably without having to perform a full QR decomposition. Second, after obtaining the sensitivity bounds, we need a method to sample elements from the dataset with as little

computational overhead as possible, ideally in only one row by row pass over the data. In this chapter, we will explore ways to deal with both of these challenges and derive two algorithms, that are suitable for real world application on large datasets.

4.1 A Fast Two-Pass Algorithm

The two-pass algorithm that we derive first is essentially made up of two components: In the first row-by-row pass over the dataset, a fast approximation of the leverage scores is computed, which is then used to sample the elements of our coreset in a second pass over the data. But before we get into the topic of efficiently approximating the leverage scores, we first cover the sampling component of the two-pass algorithm.

4.1.1 One-Pass Sampling with a Reservoir

Let's assume for a moment, that we already have the sensitivity bounds $s_1, ..., s_n$ available and that we are now interested in independently sampling k elements from our dataset $\mathcal{D} = \{(x_1, y_i)\}_{i=1}^n$, such that the i-th element has a probability of $p_i = \frac{s_i}{\sum_{j=1}^n s_j}$ of being sampled. Luckily, there already exist multiple different algorithms that solve exactly this problem in only one pass over the dataset, i.e. by only looking at each element in $\mathcal{D}$ once. One of these algorithms, that we will use as the second component of our efficient two pass coreset algorithm, is the so-called reservoir sampler by Chao [Chao, 1982].

As the name suggests, the reservoir sampler consists of a reservoir, i.e. a storage of size k, where the resulting sample will be stored. In the beginning of the sampling procedure, the reservoir is empty. Next, the algorithm decides for each element of $\mathcal{D}$, if it is added to the reservoir or not. In the first k steps of the procedure, when there is still room in the reservoir, every item is added and the reservoir is filled. After that, when the reservoir is full and there are still elements left in $\mathcal{D}$, the algorithm has to decide for each new element, if it should be added to the reservoir, and if yes, which element of the reservoir it should replace. These two decisions are the main ingredients of the algorithm, but as shown in [Chao, 1982], they turn out to be relatively simple rules.

In order to decide, if a new sample with sampling weight s_j should be included in the reservoir, the algorithm maintains at each step the sum $S_j = \sum_{l=1}^j s_l$. The decision, whether the new element is included, is then based on sampling a uniformly distributed number $q \sim U(0, 1)$. If $q \geq \frac{k \cdot s_j}{S_j}$, the new element is included, otherwise it is ignored. In case the element is included, the algorithm still has to decide, which element has to be released from the reservoir. But this decision also turns out to be simple: It suffices, to just select an element from the reservoir at random and replace it with the new element. As shown in [Chao, 1982], both of these rules together ensure, that after one pass over the entire dataset, the reservoir contains the desired sample.

In order to use this algorithm for our purposes, there is one little adjustment that we have to make. The reservoir sampler that we just described samples the elements without replacement, but one little subtlety of the sensitivity framework is that the elements are actually sampled with replacement. It turns out, that this difference can easily be overcome: Instead of using a single reservoir sampler with a reservoir of size k,

we can use k independent reservoir samplers, where each instance has a reservoir of size 1. Every element of $\mathcal{D}$ is then fed into all the k instances, and in the end we obtain our sample from the k reservoirs. This way, we can simulate sampling with replacement.

Having now found a solution to the problem of efficiently sampling elements from the dataset in only one pass, we can now turn our attention to the other problem of efficiently computing the leverage scores.

4.1.2 Fast Approximation of Statistical Leverage Scores

In order to avoid a full QR decomposition of the matrix $\sqrt{D_w}Z$, which is expensive and needs $O(nd^2)$ time, we use a method described in [Drineas et al., 2012] and improved in [Clarkson and Woodruff, 2017] to obtain approximations of the leverage scores in an efficient manner.

The idea behind this procedure is, that we first transform $\sqrt{D_w}Z$ into a much smaller matrix $\tilde{Z} \in \mathbb{R}^{t \times d}$ and then obtain the QR decomposition $\tilde{Z} = \tilde{Q}\tilde{R}$, which now only takes $O(td^2)$ time, depending on the reduced size t. Using the matrix $\tilde{R}$ of the reduced QR decomposition, we can approximate the leverage scores by computing the squared row norms of the matrix $\sqrt{D_w}Z\tilde{R}^{-1}$ (although we will later see, how we can also speed up this step). The obvious question about this procedure is, how we can obtain the reduced matrix $\tilde{Z}$, and which criteria the reduction method must satisfy in order for this idea to work.

Efficient Subspace Embeddings To acquire $\tilde{Z}$, the authors of [Clarkson and Woodruff, 2017] construct a so-called *subspace embedding*. In order to understand, what that means, suppose that we have an arbitrary matrix $A \in \mathbb{R}^{n \times d}$. When talking about a subspace embedding, we are referring to a matrix $S \in \mathbb{R}^{t \times n}$, such that

$$(1 - \epsilon)\|Ax\|_2 \leq \|SAx\|_2 \leq (1 + \epsilon)\|Ax\|_2$$

for every $x \in \mathbb{R}^d$ simultaneously and $\epsilon > 0$. This equation has a profound meaning: When viewing A as a collection of d column-vectors in $\mathbb{R}^n$, each of these vectors gets mapped into a lower dimensional subspace of $\mathbb{R}^t$, but all the distances between the original vectors as well as their lengths are preserved. For example, by choosing $x = (1, 0, ..., 0)^T \in \mathbb{R}^d$, we can see that the norm of the first column vector of A is preserved in the t-dimensional subspace up to a factor of $(1 \pm \epsilon)$. Likewise, by choosing $x = (1, -1, 0, ..., 0)^T \in \mathbb{R}^d$, we can also see that the distance between the first two column vectors is preserved in the lower dimensional space as well. By this logic, we can see that not only are all the lengths and distances of the original vectors preserved in the subspace, but also the lengths and distances between every possible linear combination of the original column vectors. In this sense, the whole column space of A is embedded into a lower dimensional subspace.

The question now is, how to choose the embedding matrix S in such a way, that SA can be computed efficiently and that the reduced size t is sufficiently small. As a solution to this problem, the authors of [Clarkson and Woodruff, 2017] developed an

efficient procedure, which not only makes it possible to construct subspace embedding matrices obliviously from the data at hand, but also enables us to compute the product SA in time of only $O(\text{nnz}(A))$, i.e. the number of non-zero entries in A. Further, the authors showed that a reduction size of $t \in O(d^2)$ is already enough to obtain a subspace embedding for our purposes of approximating the leverage scores, which is independent of the number of data points n and can be considered a huge advantage.

It is quite a surprise, that the suggested procedure can easily be described in only two simple steps: First, an all zero matrix $\tilde{A} \in \mathbb{R}^{d^2 \times d}$ is initialized, where the result of the multiplication SA will be stored. Next, each row of A is first multiplied by $+1$ or -1 with equal probability and then randomly added to one of the rows of $\tilde{A}$, also with equal probability. It turns out, that this simple procedure already yields the result of $\tilde{A} = SA$ for an embedding matrix S which represents the procedure, and that $\tilde{A}$ can be computed not only in time $O(\text{nnz}(A))$, but also in a single row by row pass over the matrix A.

Approximating the Leverage Scores The first step of the fast leverage score approximation procedure introduced in [Drineas et al., 2012] is to apply such an embedding procedure to the matrix of interest, in our case $\sqrt{D_w}Z$, which yields a matrix $\tilde{Z} \in \mathbb{R}^{d^2 \times d}$. This can be done in one row by row pass over the data by using the subspace embedding of [Clarkson and Woodruff, 2017], that we described above. In the next step, we perform a QR decomposition $\tilde{Z} = \tilde{Q}\tilde{R}$, which now takes $O(d^4)$ time. The resulting matrix $\tilde{R}$ can then be used to approximate the leverage scores in a second pass over the data by computing $\tilde{\ell}_i = \|\sqrt{w_i} z_i \tilde{R}^{-1}\|_2^2$, where $\sqrt{w_i} z_i$ is the i-th row-vector of $\sqrt{D_w}Z$.

The computation of $\tilde{\ell}_i$ takes time $O(d^2)$, but the authors of [Drineas et al., 2012] suggest another procedure to speed up this time to $O(d \log(n))$, which can be useful if $d > \log(n)$. Their idea is to apply a so-called Johnson-Lindenstrauss transformation [Johnson and Lindenstrauss, 1984] to R^{-1}, in order to reduce its size to only $d \times m$ elements, where $m \in O(\log(n))$. What this means is that the matrix $R^{-1} \in \mathbb{R}^{d \times d}$ is multiplied by a random matrix $G \in \mathbb{R}^{d \times m}$, where each entry of G follows a normal distribution with mean zero and variance $\frac{1}{m}$, i.e. $G_{ij} \sim \mathcal{N}(0, \frac{1}{m})$. The resulting product $R^{-1}G$ can be computed once in the beginning, and then for every single row it takes only $O(d \log(n))$ time to compute the approximated leverage score $\tilde{\ell}_i = \|\sqrt{w_i} z_i (R^{-1}G)\|_2^2$. It was shown in [Drineas et al., 2012], that the approximations $\tilde{\ell}_i$ of the leverage scores satisfy that

$$(1 - \epsilon)\ell_i \leq \tilde{\ell}_i \leq (1 + \epsilon)\ell_i$$

for $\epsilon > 0$, where ℓ_i is the true leverage score. We thus have obtained a constant factor approximation of the true leverage scores that can be computed efficiently in only two passes over the data and we can argue, that the constant factor approximation of the leverage scores doesn't affect the asymptotic analysis of the sensitivity framework (see theorem 4). Thus, we can replace the true leverage scores with the approximated leverage scores in our sampling distribution without having any impact on the coreset size.

4.1.3 Putting it all together

We can combine the idea of reservoir sampling with the fast approximation method of the statistical leverage scores to improve the running time of the naïve algorithm in section 3.3.4, so that it now only takes two passes over the data. The resulting new algorithm is given in Algorithm 3.

In a first pass, the subspace embedding of [Clarkson and Woodruff, 2017] is applied to the reweighted scaled model matrix $\sqrt{D_w}Z$, which takes $O(\text{nnz}(Z))$ computation time. The subsequent QR decomposition of the reduced $d^2 \times d$ matrix then runs in time $O(d^4)$ and the computation of $\tilde{R}^{-1}$ as well as the Johnson Lindenstrauss transformation take $O(d^3)$ time, because the computation is dominated by the matrix inversion.

In the second pass, every row is fed into each of the k independent reservoir samplers, which each have a reservoir of size one. This ensures, that the samples are drawn with replacement. If the Johnson Lindenstrauss transformation was applied, the second pass runs in time $O(\text{nnz}(Z)\log(n))$. The resulting coreset is then simply the content of the k reservoirs of the independent reservoir samplers and the total running time of the algorithm is $O(\text{nnz}(Z)\log(n) + \text{poly}(d))$, which is dominated by the second pass over the data.

4.2 A One-Pass Online Algorithm

The two-pass algorithm is a fast and practical procedure to construct coresets in the context of probit regression in most situations. But sometimes, situations arise where two passes over the dataset are just not feasible. For example, what if the data arrives in real time, with many thousand samples per second, and we simply don't have enough storage to keep all the records? In such situations, we need to make our sampling decisions immediately: Do we include a new sample in the coreset, or do we discard it forever? For these situations, we need to come up with new algorithmic ideas, because the two-pass sampling of our previous algorithm is simply not up to the task.

The reason, why Algorithm 3 needs two passes, is that it first computes a fast approximation of the leverage scores in the first pass, and then obtains a sample in the second pass. If we want to do both, approximating the leverage scores as well as obtaining the sample, we have to find a way to approximate the leverage scores in an online manner, i.e. when the data arrives row by row and we don't have any knowledge of the data that is about to come in the form of future samples. In this work, we follow a train of thought by the authors of [Cohen et al., 2020], who investigated how the statistical leverage scores can be approximated in such an online scenario, where the data arrives row by row and a sampling decision has to be made immediately.

The first, and most important observation by the authors of [Cohen et al., 2020] is, that we can easily obtain overestimates of the leverage scores of a matrix, if we remove some of its rows. To be more specific, consider the matrix $A \in \mathbb{R}^{n\times d}$ and remember that the i'th leverage score is given by $\ell_i = a_i^T(A^TA)^{-1}a_i$, where the vector $a_i \in \mathbb{R}^d$ represents the i'th row of A. Now, imagine that the matrix A_j only contains the first j rows of A, and that we use A_j to compute the approximation $\hat{\ell}_i = a_i^T(A_j^TA_j)^{-1}a_i$. In [Cohen et al.,

Algorithm 3: Fast two-pass algorithm for coreset construction

Input: Dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ with weight vector $w \in \mathbb{R}_{>0}^n$, $\mathcal{W} = \sum_{i=1}^n w_i$, size parameter $k \in \mathbb{N}$

Output: A subset $\mathcal{C} \subseteq \mathcal{D}$ of size $|\mathcal{C}| = k$ with weight vector $u \in \mathbb{R}_{>0}^k$

Initialize $\tilde{Z} = 0 \in \mathbb{R}^{d^2 \times d}$
for $i = 1, ..., n$ **do** `// first pass`
  $z_i = -\sqrt{w_i}(2y_i - 1)x_i^T$ `// row vectors of` $\sqrt{D_w}Z$
  j = random sample from $\{1, ..., d^2\}$ with equal probability
  l = random sample from $\{+1, -1\}$ with equal probability
  $\tilde{Z}_j = \tilde{Z}_j + l \cdot z_i$ `// update the` j`'th row of` $\tilde{Z}$
Compute the QR decomposition of $\tilde{Z} = \tilde{Q}\tilde{R}$
Initialize $G = I \in \mathbb{R}^{d \times d}$
if $\lceil \log(n) \rceil < d$ **then**
  $G = 0 \in \mathbb{R}^{d \times \lceil \log(n) \rceil}$
  Draw $G_{ij} \sim \mathcal{N}(0, \frac{1}{\lceil \log n \rceil})$ `// draw Johnson-Lindenstrauss matrix`
Compute $M = \tilde{R}^{-1}G$
Initialize $u = 0 \in \mathbb{R}^k$ `// empty weight vector`
Initialize k independent weighted size-1-reservoir samplers $S_1, ..., S_k$
for $i = 1, ..., n$ **do** `// second pass`
  $\tilde{\ell}_i = \|z_i M\|_2^2$ `// approximate leverage scores`
  $a_i = \tilde{\ell}_i + \frac{w_i}{\mathcal{W}}$ `// sensitiviy bound`
  $s_i = w_i 2^{\lceil \log(\frac{a_i}{w_i}) \rceil}$ `// rounding to control VC dimension`
  for $j = 1, ..., k$ **do**
    Feed (x_i, y_i) with sampling weight s_i to S_j
    if S_j samples (x_i, y_i) **then**
      $u_j = \frac{w_j}{s_i k}$ `// unnormalized weights`
$\mathcal{C}$:= elements from the k reservoirs
$u = u \cdot \sum_{i=1}^n s_i$ `// normalize weights`
return $\mathcal{C}$, u

2020] it was shown, that in this case we always have that $\hat{\ell}_i \geq \ell_i$, i.e. our approximation $\hat{\ell}_i$ that was obtained by only considering the first j rows of A, is an upper bound for the true leverage score ℓ_i.

At this point, we have to remind ourselves of the core theorem of the sensitivity framework (theorem 4), which forms the basis of all our coreset construction endeavors. In this theorem, the most critical aspect is to find upper bounds on the sensitivity scores in order to obtain a sampling distribution that can yield a coreset, and we accomplished this by showing, that the statistical leverage scores are upper bounds on the sensitivities for probit regression. Now, consider the overestimates $\hat{\ell}_i$. Obviously, because $\hat{\ell}_i \geq \ell_i$, these overestimates could also be used in our sampling distribution. As long as the sum

of these overestimates is small enough, the resulting sample will also be a coreset.

4.2.1 A Naïve Online Algorithm

We now have the basis for our first naïve online algorithm. For a d-dimensional dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ that arrives in an online manner, we can at every step i maintain a matrix $M^{(i)} = M^{(i-1)} + w_i x_i x_i^T$, $M^{(0)} = 0 \in \mathbb{R}^{d \times d}$ that represents the sum of the outer products of the first i weighted rows, much like the matrix $A_j^T A_j$ above. Then, we can obtain overestimates of the statistical leverage scores by computing $\hat{\ell}_i = w_i x_i^T (M^{(i)})^\dagger x_i$, where the † symbol denotes the Moore-Penrose pseudoinverse. After one pass, the resulting scores $\hat{\ell}_1, ..., \hat{\ell}_n$ are all overestimates of the true scores of the weighted matrix $\sqrt{D_w} Z$, that we needed for our original sampling distribution. The pseudocode for this naïve algorithm is given in Algorithm 4.

In order to prove, that the naïve online algorithm does in fact construct small coresets, we need to show that the sum of the overestimates $\hat{\ell}_1, ..., \hat{\ell}_n$ is small, because this sum directly impacts the coreset size in the sensitivity framework. Luckily, this work has already been done by the authors of [Chhaya et al., 2020] in an effort to build on the work by [Cohen et al., 2020].

Lemma 9 ([Chhaya et al., 2020])**.** *Let $A \in \mathbb{R}^{n \times d}$ and let $\hat{\ell}_1, ..., \hat{\ell}_n$ be overestimates of the statistical leverage scores of A that were obtained by computing $\hat{\ell}_i = a_i^T (A_i^T A_i)^\dagger a_i$, where A_i is the matrix that only consists of the first i rows of A and the vector $a_i \in \mathbb{R}^d$ represents the the i'th row of A. Then, it holds that*

$$\sum_{i=1}^{n} \hat{\ell}_i \in O(d + d \log \|A\|_2),$$

where $\|A\|_2$ is the spectral norm of A, i.e. $\|A\|_2 = \sigma_{max}(A)$, where $\sigma_{max}(A)$ is the largest singular value of A.

We can directly use this result in order to show that the naïve algorithm is correct and indeed yields a small coreset by only passing once over the dataset.

Theorem 6. *Algorithm 4 returns a $(1 \pm \epsilon)$-coreset for probit regression, if the size parameter k satisfies*

$$k \in O\left(\frac{\mu d^2 \log(\sigma_{max})}{\epsilon^2} \log(\mu \omega n) \log(\mu d \log(\sigma_{max})) \right),$$

where σ_{max} is the largest singular value of $\sqrt{D_w} Z$.

Proof. As a result of lemma 9, the total sum of the sensitivity bounds is now

$$S \in O(\mu d + \mu d \log(\sigma_{max})) \subseteq O(\mu d \log(\sigma_{max})),$$

where $\sigma_{max} = \|\sqrt{D_w} Z\|_2$ is the largest singular value of $\sqrt{D_w} Z$. We still round our sampling weights $\hat{\ell}_i + \frac{w_i}{\mathcal{W}}$ to keep control of the VC dimension, which thus remains

Algorithm 4: Naïve online algorithm for coreset construction

Input: Dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ with weight vector $w \in \mathbb{R}^n_{>0}$, $\mathcal{W} = \sum_{i=1}^n w_i$, size parameter $k \in \mathbb{N}$

Output: A subset $\mathcal{C} \subseteq \mathcal{D}$ of size $|\mathcal{C}| = k$ with weight vector $u \in \mathbb{R}^k_{>0}$

Initialize $M^{(0)} = 0 \in \mathbb{R}^{d \times d}$

Initialize k independent weighted size-1-reservoir samplers $S_1, ..., S_k$

for $i = 1, ..., n$ **do**

- $z_i = -\sqrt{w_i}(2y_i - 1)x_i^T$ `// row vectors of` $\sqrt{D_w}Z$
- $M^{(i)} = M^{(i-1)} + z_i^T z_i$ `// rank one update`
- $\hat{\ell}_i = \min\{z_i(M^{(i)})^\dagger z_i^T, 1\}$ `// approximate leverage scores`
- $a_i = \hat{\ell}_i + \frac{w_i}{\mathcal{W}}$ `// sensitivity bound`
- $s_i = w_i 2^{\lceil \log(\frac{a_i}{w_i}) \rceil}$ `// rounding to control VC dimension`
- **for** $j = 1, ..., k$ **do**
  - Feed (x_i, y_i) with sampling weight s_i to S_j
  - **if** S_j samples (x_i, y_i) **then**
    - $u_j = \frac{w_j}{s_i k}$ `// unnormalized weights`

$\mathcal{C} :=$ elements from the k reservoirs

$u = u \cdot \sum_{i=1}^n s_i$ `// normalize weights`

return $\mathcal{C}$, u

unchanged at $\Delta \in O(d \log(\mu\omega n))$, where $\omega = \frac{w_{max}}{w_{min}}$ is the ratio of the largest and smallest weight. We can plug everything into the main theorem of the sensitivity framework by setting the failure probability equal to $\delta = n^{-c}$ for any absolute constant $c > 1$:

$$\begin{aligned} k &\in O\left(\frac{S}{\epsilon^2}\left(\Delta \log S + \log\left(\frac{1}{\delta}\right)\right)\right) \\ &\subseteq O\left(\frac{\mu d \log(\sigma_{max})}{\epsilon^2}\left(d \log(\mu\omega n) \log(\mu d \log(\sigma_{max})) + \log(n^c)\right)\right) \\ &\subseteq O\left(\frac{\mu d^2 \log(\sigma_{max})}{\epsilon^2} \log(\mu\omega n) \log(\mu d \log(\sigma_{max}))\right) \end{aligned}$$

□

4.2.2 Improving the Naïve Algorithm

There is one issue with the naïve online algorithm, that calls for improvement. Although it requires only a single pass over the dataset, it needs to compute the pseudoinverse of M for every new datapoint, which takes $O(d^3)$ time. Thus, the overall running time of the algorithm is $O(nd^3)$, which is even slower than conducting a full QR decomposition.

Fortunately, the authors of [Chhaya et al., 2020] came up with a clever idea of how the running time can be improved. Instead of computing the pseudoinverse over and

over again for each new datapoint, they make use of a variant of the so-called Sherman-Morrison formula [Sherman and Morrison, 1950], which allows for updating an already computed pseudoinverse more efficiently.

Lemma 10 ([Chhaya et al., 2020]). *Let $M \in \mathbb{R}^{d\times d}$ and let $x \in \mathbb{R}^d$ be in the columnspace of M. Then,*

$$(M + xx^T)^\dagger = M^\dagger - \frac{M^\dagger x x^T M^\dagger}{1 + x^T M^\dagger x},$$

where $M^\dagger$ denotes the pseudoinverse of M.

If we already have $M^\dagger$ available, this formula allows us to compute the pseudoinverse of $M + xx^T$ in only $O(d^2)$ time, provided that x is in the column space of M. In order to check, if x is indeed in the column space of M, the authors of [Chhaya et al., 2020] suggest to maintain an orthonormal basis Q of the columnspace of M and then to check, whether $\|Qx\|_2 = \|x\|_2$. When this is the case, x is in the columnspace of Q, and thus it is also in the columnspace of M and the adapted Sherman-Morrison formula can be applied.

We can use these findings to improve our naïve procedure as follows: At every step of the iteration, we maintain the current values of M, $M^\dagger$ and Q. At every time, when x is in the columnspace of Q, we use the adapted Sherman-Morrison formula to update $M^\dagger$, which only takes $O(d^2)$ time. If x is not in the columnspace of Q, we can just compute the Moore-Penrose pseudoinverse, which takes $O(d^3)$ time, but because M is a $d \times d$ matrix, this case can happen only a maximum of d times. Thus, the total running time of the procedure reduces to an amortized time of $O(nd^2)$. To conclude this section on efficient coreset algorithms, we present the improved online algorithm in Algorithm 5.

5 Experiments

It is now time to investigate, how the two efficient algorithms that we derived in the previous section behave on real world datasets, both in the context of maximum likelihood estimation as well as in a Bayesian setting. In order to do so, we selected three benchmark datasets in advance that will be used to compare both algorithms to each other as well as to the baseline uniform sampling procedure, where each sample is simply selected with equal probability.

It is important to emphasize, that the datasets used in the evaluation are not a result of cherry picking: We selected the same datasets as [Munteanu et al., 2018] as well as [Mai et al., 2021] in order to be comparable to other works in the field, without testing in advance if our algorithms look good on them or not.

Finally, we note that all of the following experiments were implemented in Python and executed on an AMD Ryzen 7 2700x processor with 8 cores of 3.7GHz clock speed and 16GB of RAM. The code for all the experiments is open source and can be found publicly accessible on Github.[1]

[1] `https://github.com/cxan96/efficient-probit-regression`

Algorithm 5: Online algorithm for coreset construction

Input: Dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ with weight vector $w \in \mathbb{R}^n_{>0}$, $\mathcal{W} = \sum_{i=1}^n w_i$, size parameter $k \in \mathbb{N}$

Output: A subset $\mathcal{C} \subseteq \mathcal{D}$ of size $|\mathcal{C}| = k$ with weight vector $u \in \mathbb{R}^k_{>0}$

Initialize $M = M_{inv} = Q = 0 \in \mathbb{R}^{d \times d}$
Initialize k independent weighted size-1-reservoir samplers $S_1, ..., S_k$
for $i = 1, ..., n$ **do**
  $z_i = -\sqrt{w_i}(2y_i - 1)x_i^T$ // row vectors of $\sqrt{D_w}Z$
  $M = M + z_i^T z_i$ // rank one update
  if $\|Qz_i^T\|_2 = \|z_i\|_2$ **then** // z_i^T in column space of Q?
    $M_{inv} = M_{inv} - \frac{M_{inv} z_i^T z_i M_{inv}}{1 + z_i M_{inv} z_i^T}$ // adapted Sherman-Morrison formula
  else
    $M_{inv} = M^\dagger$ // Moore-Penrose pseudoinverse
    $QR = M$ // QR decomposition of M
  $\ell_i = \min\{z_i M_{inv} z_i^T, 1\}$ // approximate leverage scores
  $a_i = \ell_i + \frac{w_i}{\mathcal{W}}$ // sensitivity bound
  $s_i = w_i 2^{\lceil \log(\frac{a_i}{w_i}) \rceil}$ // rounding to control VC dimension
  for $j = 1, ..., k$ **do**
    Feed (x_i, y_i) with sampling weight s_i to S_j
    if S_j samples (x_i, y_i) **then**
      $u_j = \frac{w_j}{s_i k}$ // unnormalized weights
$\mathcal{C} :=$ elements from the k reservoirs
$u = u \cdot \sum_{i=1}^n s_i$ // normalize weights
return $\mathcal{C}$, u

5.1 Datasets

The datasets that are used in the empirical evaluation of our algorithms are called Covertype[2], Kddcup[3] and Webspam[4] and are all publicly available.

The Covertype dataset consists of $n = 581012$ observations of 30x30m forest patches of wilderness areas located in the Roosevelt National Forest of Northern Colorado, USA. The task is to predict the so called covertype of each of these patches, i.e. the dominant tree species, based on $d = 54$ observed features. There are seven distinct tree species that appear in the dataset, so we have a multiclass problem. To transform the problem into a binary classification problem that can be subjected to a probit analysis, we adapted the task to distinguish the tree species "Lodgepole Pine" from the other six species, thus obtaining a balanced problem of 49% positive vs 51% negative observations. As an

[2] https://archive.ics.uci.edu/ml/datasets/Covertype
[3] https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
[4] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#webspam

additional preprocessing step, all continuous features of the dataset were scaled to have a mean of zero as well as unit variance. In addition, an intercept column of all ones was appended to the data.

The Webspam datasets consists of $n = 350000$ observations of web pages, that can either be classified as spam or no spam, based on the occurence of 254 distinct words on the web page. 61% of the observations in the dataset are labeled as spam and the other 39% are labeled as no spam. The task is to predict, whether a given web page is spam or not. The features consist of binary 0/1 observations that indicate, if a given word is present on a web page or not. In a preprocessing step, words that are never present in the dataset were removed, as well as words that are only present on a single web page. After preprocessing, the dataset that is subjected to the experiments consists of $d = 128$ binary features. An intercept column of all ones was appended to the Webspam dataset as well.

The Kddcup dataset consists of $n = 494021$ observations of network connections, where the task is to predict if a connection is "good" or "bad", i.e. to distinguish, whether a hacker tried to gain unauthorized access to a network or whether someone tried to establish a normal and authorized connection. 80% of the connections in the dataset are "bad" and the other 20% are "good" connections. In a preprocessing step, $d = 33$ continuous features were retained from the data and scaled to a mean of zero and unit variance. Just like the other datasets, an intercept column of all ones was appended to the Kddcup data as well.

5.1.1 A Visual Comparison of the Datasets

Before we evaluate the three competing algorithms on each of the datasets, we first make an attempt at uncovering some of the structural characteristics of the data, that could potentially help us to learn more about the situations in which the algorithms perform well or might fail.

In order to do so, we first project each dataset onto its first two principal components, which where obtained as a result of a principal component analysis [Jolliffe, 2002]. The first two principal components are orthogonal vectors that represent the two directions of the data, in which the highest variance occurs. By projecting each datapoint onto the subspace spanned by the first two principal components, we have a way of visualizing the datasets in a two dimensional space, while still retaining as much variance as possible, even though we are reducing the dimensionality to only 2.

Next, we draw two distinct samples from the reduced datapoints, each consisting of 500 points. The first one is a uniform sample, where each datapoint is sampled with equal probability. The second sample is drawn with probabilities that are proportional to the statistical leverage scores of the observations with respect to the original dataset, without applying a PCA. Both of these samples are presented for each of the datasets in figure 3.

We can see, that the samples reveal some interesting characteristics of the datasets. Starting with Covertype, we notice that there hardly seems to be any difference between the uniform sample and the leverage score sample. Taking into consideration, that the

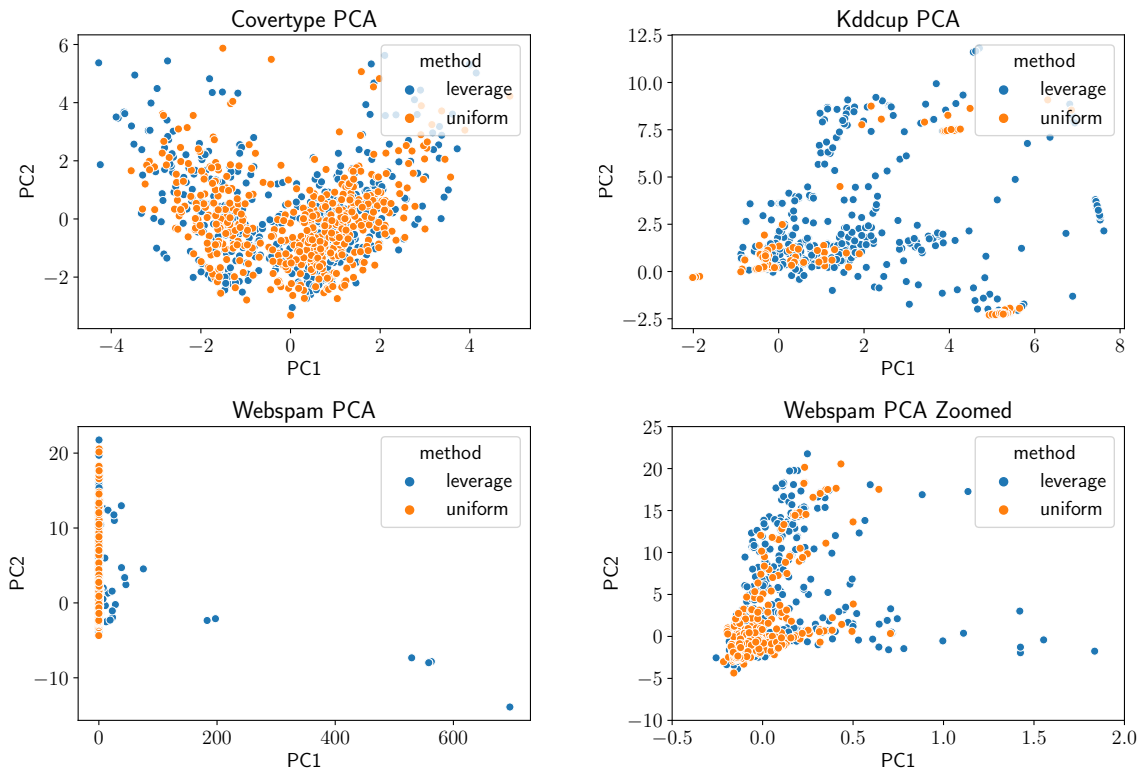

Figure 3: The datasets are compared by projecting the datapoints onto the first two principal components and then drawing a random sample of 500 points (a) uniformly and (b) proportionally to the statistical leverage scores of the original data.

statistical leverage scores assign a higher importance to outliers or "unusual" datapoints, we can conclude that there don't seem to be too many unusual observations within the Covertype dataset, or else the two samples would differ more substantially. It seems to be the case, that a uniform sample is already sufficient to capture most of the inherent structure of the data, without having to put extra weight on unusual datapoints, which simply appear not to exist in this dataset. Thus, from now on, it seems reasonable to think of the Covertype dataset as being a representative for such situations, where the data is already quite uniformly distributed and not a lot of outliers are present, i.e. where a uniform sample already seems to yield good results.

In contrast to Covertype, the Kddcup dataset exhibits a different picture. Here, we can see that the uniform sample and the leverage score sample differ way more substantially than it was the case for the Covertype dataset. While the uniform sample only seems to occupy a very limited portion of the feature space, most of it being a cluster of many points close to the origin, the leverage score sample covers a lot more of the feature space, indicating that there are a variety of unusual datapoints, or outliers present, which the uniform sample misses. Thus, it seems that we can think of the Kddcup dataset as an example of a situation, where on the one hand, we have a lot of similar observations that are concentrated within a comparatively small proportion of the feature space, but on the other hand, we also have a considerable amount of unique observations, which

are scattered around the feature space while seemingly not belonging to any particular cluster.

Lastly, another interesting situation is presented to us when taking a look at the two samples of the Webspam dataset. What strikes the eye first, is that there seem to be some truly extreme outliers present in the dataset, that the leverage score sampling picked up on. Only by zooming in on the more crowded area of the feature space and thereby ignoring those extreme outliers, we can see the remaining parts of the two samples, which seem to be rather similar, just like in the situation of the Covertype dataset. It thus seems to be the case, that the Webspam dataset is an example of a situation, where on the one hand, the vast majority of the datapoints are concentrated around some region of the feature space, but on the other hand, there are also a few hard hitting outliers present, which exhibit enormous differences to the majority of the other observations.

5.2 Coreset-Based Maximum Likelihood Estimation

We are now ready to investigate, how well our three algorithms (two pass coreset construction, online coreset construction and uniform sampling) perform at the task of data reduction with the goal of estimating the parameter vector of a probit model on the three datasets that we just introduced via maximum likelihood estimation. In order to do so, we present the following experimental setup:

For each of the datasets, we first obtain the objective function $f(\beta)$ of the original optimization problem without applying any data reduction and then solve the problem to find the unique solution β^{opt}. Next, we apply our data reduction algorithms in order to select a small subset of the original data, yielding the reduced objective function $\tilde{f}(\beta)$. We solve this smaller optimization problem in order to obtain the solution $\tilde{\beta}$. Our goal is to see, if the solution on the reduced dataset, $\tilde{\beta}$, is also a good solution for the original problem. In order to evaluate the approximation quality, we compute what we call the approximation ratio:

$$\text{approximation ratio} = \frac{f(\tilde{\beta})}{f(\beta^{opt})},$$

which always evaluates to a real number in the interval $[1, \infty)$. The closer this value is to 1, the better is the quality of the approximation.

We run this procedure for each algorithm on each of the datasets for multiple different reduction sizes. Because the algorithms are not deterministic, we ran each of the experiments a total of 51 times. The resulting medians as well as the normalized inter quartile range can be seen in figure 4.

5.2.1 Comparison of Approximation Quality

Starting with the Covertype dataset, we can see that each algorithm quickly reaches good approximation ratios of less than 1.02 for subset sizes of only 15000 datapoints, which are less than 3% of the original dataset. Both, the uniform sampling as well as the two pass algorithm, are close to each other in terms of approximation quality, although the median ratio of the two pass algorithm is always better than that of uniform sampling.

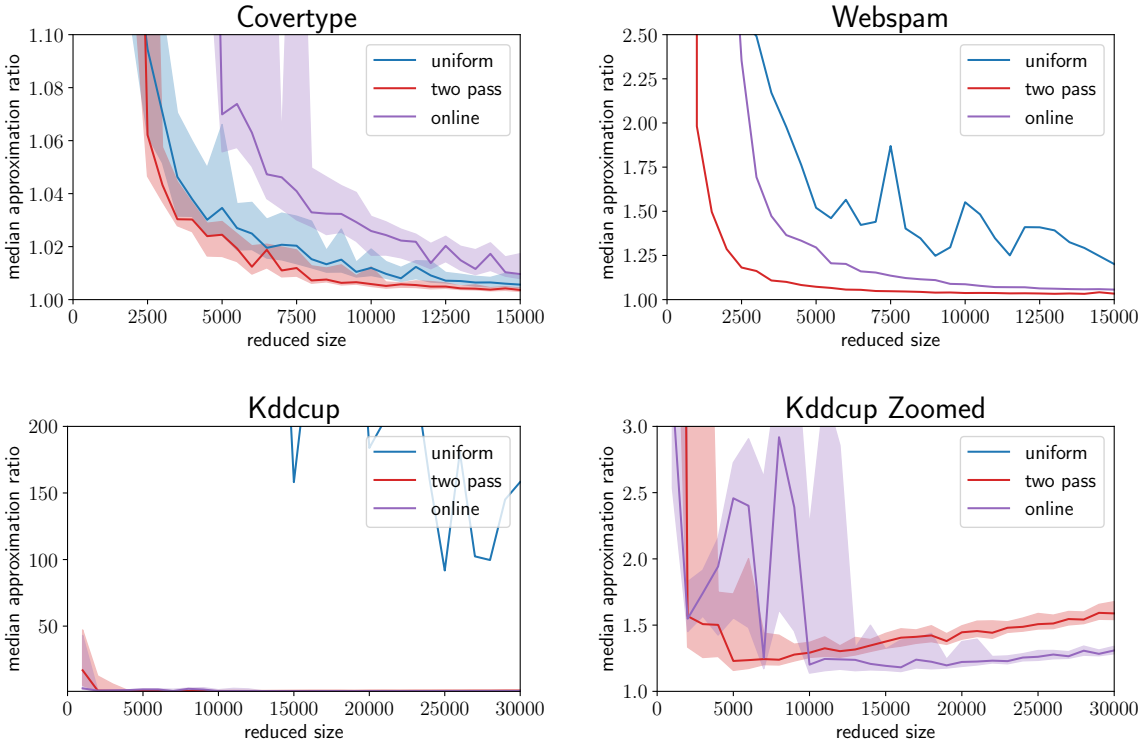

Figure 4: The medians as well as the normalized inter quartile ranges of the approximation ratios of the three algorithms uniform sampling, two pass coreset construction and online coreset construction on the datasets Covertype, Webspam and Kddcup for different data reduction sizes. For each reduction size, the experiments were repeated a total of 51 times.

The online algorithm performs worse than the two competing algorithms, but it also quickly reaches a low approximation ratio of less than 1.02. Interestingly, the fact that the results of our three algorithms don't differ much on the Covertype dataset, could be explained by our earlier findings regarding the characteristics of the data. For the Covertype dataset, we found that it could be a representative for those situations, where the datapoints are already quite homogeneously distributed, i.e. there are not a lot of outliers, heavy hitters or other unusual points to worry about and thus, a uniform sample can already be sufficient to capture the relevant characteristics of the data. Nevertheless, our two pass algorithm still outperformed the uniform sampling procedure, albeit by only a small margin.

For the Webspam dataset, the picture looks a lot different. Here, we can see for the first time that the uniform sampling algorithm fails, while the two pass algorithm as well as the one pass algorithms quickly reach low approximation ratios. On top of that, the approximation ratios of the uniform sampling algorithm exhibit a much higher degree of variation and thereby show a lot less stability than the competing algorithms two pass and online. Again, it is interesting to note, that these results could also potentially be explained by our earlier observations: When visualizing the Webspam dataset by using the first two principal components, we already noticed that there are a few extreme

outliers present in the data. We also noticed, that the uniform sample was unable to pick up on those outliers and only concentrated on the bulk of the datapoints in the center of the feature space. Now, in our maximum likelihood experiments, we can see that the uniform sampling procedure also exhibits major weaknesses for the Webspam dataset. It thus seems reasonable to assume, that this is again happening because the uniform sampling algorithm misses the extreme outliers, which could potentially have a high impact on the objective function. Both, the two pass algorithm as well as the online algorithm, don't miss those important points thanks to their more adaptive importance sampling distributions that also take the statistical leverage scores into account, and therefore show a stable convergence behavior towards low approximation rates.

The most dramatic differences between uniform sampling and the competing algorithms can be observed when looking at the results for the Kddcup dataset. Here, we can see that uniform sampling fails terribly and doesn't even reach approximation ratios lower than 50. When zooming in, we can see that our other two algorithms perform much better, although they also seem to show some difficulties. It is especially interesting to note, that the online algorithm seems to perform better than the two pass algorithm, even though its approximations of the statistical leverage scores are less accurate. Together with our earlier observations of the structure of the Kddcup dataset, it seems that Kddcup is a particularly difficult dataset to compress. Even though the sampling methods that also rely on the leverage scores clearly outperform the uniform sampling algorithm, they also show difficulties with regard to convergence. We should also note here, that the maximum size of 30000 points for the Kddcup dataset already takes up more than 6% of the whole dataset, which is more than double as much as what was needed for the Covertype dataset to achieve a much better approximation quality. Thus, the Kddcup dataset seems to be more challenging for all the competing algorithms, although the two pass algorithm and the online algorithm handle the difficult conditions way better than the uniform sampling algorithm.

5.2.2 Comparison of Running Times

We round off our first experiment with a discussion regarding the runtimes of the algorithms compared to the time it takes to solve the original problem without prior reduction, as well as a discussion of the tradeoffs between approximation quality and running time.

It must be noted, that the online coreset algorithm had to be excluded from the comparison due to its incomparably high running time of $O(nd^2)$, which would have made it impossible to execute a sufficient number of experiments. However, we remark that the advantage of the online algorithm isn't necessarily its speed, but that it is able to construct coresets even when two passes over the data are impossible and every sampling decision has to be made immediately. Also, it might be possible to drastically increase the efficiency of the online algorithm, by running mutiple parallel instances of it on a distributed cluster and then equally balancing the load of the incoming data records between each of the instances, but these possible adaptations are left open for future work.

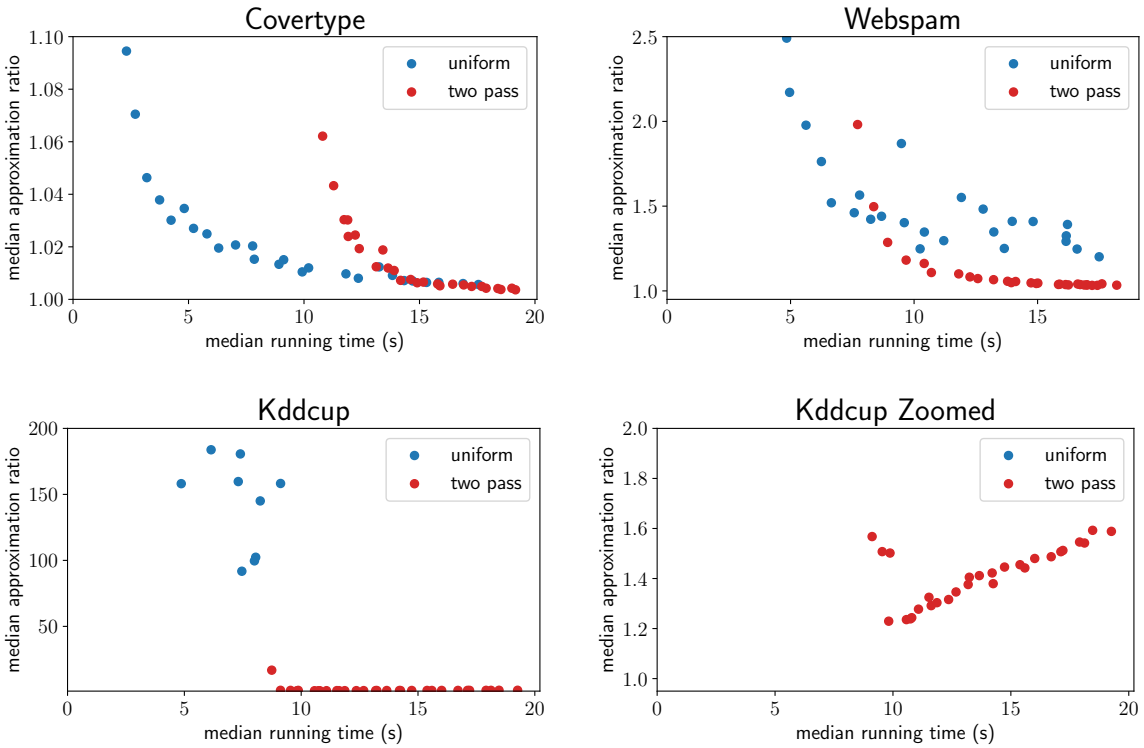

Figure 5: The plots show the tradeoffs between running time and approximation ratio for the uniform sampling algorithm and the fast two pass algorithm on our datasets. Every point represents the median in running time, as well as the median in approximation ratio, for a given data reduction size.

The total running times for obtaining the maximum likelihood estimate of the probit model for each of the datasets without applying any data reduction are given in table 1 and will serve as a baseline for further discussions.

	Covertype	**Kddcup**	**Webspam**
Total running time	535 seconds	103 seconds	447 seconds

Table 1: The total running times that it takes to obtain the maximum likelihood estimator for each of the datasets.

In figure 5, we present the tradeoffs between approximation ratio and running time for uniform sampling as well as for the fast two pass algorithm. The reported times include the data reduction step as well as the optimization step, in order to be able to compare the times to the total running times without data reduction.

When looking at the results for the Covertype dataset, we can see that the uniform sampling algorithm outperforms the two pass algorithm in terms of speed, but is unable to reach the same degree of approximation quality as the two pass algorithm. The running times of the uniform sampling procuedure range roughly between 1 and 10 seconds, but decent results are only achieved when investing at least 5 seconds. On

the other hand, the running times of the two pass algorithm range between 10 and 20 seconds, thus taking almost twice as long. But compared to the total running time for the Covertype dataset without reduction, we can see that even the two pass algorithm reduces the total running time by more than 96%, from 535 seconds to less than 20 seconds.

Taking a look at the Webspam dataset, we can see an interesting picture: Not only does the two pass algorithm show better approximation rates than the uniform sampling algorithm, but it also shows similar running times. Thus, for the Webspam dataset, the two pass algorithm almost dominates the uniform sampling in the two criterias, since it has similar running times but achieves a better approximation quality. For running times of 15 seconds, the fast two pass algorithm already achieves decent results, thus we can say that the original time for solving the unreduced problem was again reduced by more than 96%, from 447 seconds to only 15 seconds.

In the case of Kddcup, the comparison is almost pointless, because the uniform sampling fails. We only include these results to show that the running times of the two pass algorithm consistently range between 10 and 20 seconds, even when the data is substantially less well behaved, as we showed for the Kddcup dataset. Here, the time savings are not quite as impressive as for the Covertype and the Webspam dataset: The total running time was "only" reduced by more than 80%, from 103 seconds to less than 20 seconds. Still, we can consider this a decent win, especially when keeping in mind that the uniform sampling algorithm failed completely.

To conclude the discussion of our first experiment, we can say that the slightly increased running time of the two pass algorithm compared to the uniform sampling algorithm should be of no concern, because the few seconds in excess runtime were always well compensated by substantial gains in approximation quality. For each dataset, the two pass algorithm is able to reduce the total optimization time by more than 80%, even 96% for the Covertype and for the Webspam dataset.

5.3 Coreset-Based Bayesian Inference

The goal of our second experiment is to investigate, if our algorithms can be successfully applied as a data reduction step to cut down the computational cost of performing a full Bayesian probit analysis on our three datasets. In order to do so, we use the efficient Gibbs sampler described earlier in section 2.4.2 to generate samples with and without a preceding data reduction step using our algorithms, and see how they compare to each other. But before we can get into the full experimental setup, we must note that in order to use the Gibbs sampler together with our algorithms, it has to be slightly adapted to account for the sample weights that are an essential component of our coreset algorithms, which is briefly covered in the next section.

5.3.1 Adapting the Gibbs Sampler for Weighted Coresets

The first part of the Gibbs sampler that has to be slightly adapted is the expression $B = (M^{-1} + X^T X)^{-1}$, where $M \in \mathbb{R}^{d \times d}$ is the prior covariance matrix and $X \in \mathbb{R}^{n \times d}$ is

the model matrix. When using our algorithms to draw a random sample of k datapoints according to the distribution $p_1, ..., p_n$ and arranging them in a matrix $C \in \mathbb{R}^{k \times d}$, the matrix C is now random too, because it is the result of drawing the random sample of our datapoints. Our goal is now, to find weights $u_1, ..., u_n$ for our data points, such that when C is a random sample of the reweighted datapoints $u_1 x_1, ..., u_n x_n$, we have

$$E[C^T C] \stackrel{!}{=} X^T X,$$

where X is the model matrix of our original unweighted datapoints. We can accomplish this by calculating the expected value as follows:

$$\begin{aligned} E[C^T C] &= E\left[\sum_{j=1}^{k} c_j c_j^T\right] = \sum_{j=1}^{k}\sum_{i=1}^{n} u_i^2 p_i x_i x_i^T \\ &= k\sum_{i=1}^{n} u_i^2 p_i x_i x_i^T \stackrel{!}{=} \sum_{i=1}^{n} x_i x_i^T = X^T X. \end{aligned}$$

By comparing the coefficients, we see that $u_i = \frac{1}{\sqrt{k p_i}}$ is the required weight. Thus, when the Gibbs sampler is applied to a subset $\mathcal{C}$ of our initial dataset, we need to multiply every datapoint of the subset with the weight $u_i = \frac{1}{\sqrt{k p_i}}$, before we can plug it into the expression for obtaining B.

Next, we need to adapt the expression $X^T y^*$, where y^* is the current vector of the latent variables. We do this in a similar fashion, by asking how we should reweight the original datapoints as well as the latent variables in order to be unbiased, i.e. we want that

$$E\left[C^T y_C^*\right] = X^T y^*,$$

where y_C^* is the random sample of the latent variables that is associated with the random sample in C. Luckily, it turns out, that the weights that we obtained earlier are also correct for adapting this expression:

$$E\left[C^T y_C^*\right] = E\left[\sum_{j=1}^{k} c_j^T (y_C^*)_j\right] = \sum_{j=1}^{k}\sum_{i=1}^{n} u_i^2 p_i x_i^T y_i^* = k\sum_{i=1}^{n}\left(\frac{1}{\sqrt{k p_i}}\right)^2 p_i x_i^T y_i^* = X^T y^*.$$

It follows, that when using the Gibbs sampler in conjunction with our algorithms, we have to reweight our sample with weights $u_i = \frac{1}{\sqrt{k p_i}}$ both when obtaining $X^T X$ as well as $X^T y^*$.

5.3.2 Experimental Setup

We used a similar experimental setup as in [Huggins et al., 2016] and [Geppert et al., 2017]: First, we arbitrarily selected a relatively uninformative prior distribution of

$$\beta \sim \mathcal{N}(0, 10 \cdot \mathcal{I}),$$

where $\mathcal{I}$ is the identity matrix. Next, we used the Gibbs sampler to generate a sample of the full posterior distribution without applying any data reduction for each of the three datasets. Each of the three full posterior samples has a size of 10000, and for each of the datasets, a different burn in value was selected. For Covertype, a burn in of 200 was sufficient, 2000 was needed for Kddcup and 3000 for Webspam.

Next, we applied our algorithms to each of the datasets for a variety of reduction sizes and ran the adapted version of the Gibbs sampler (see Section 5.3.1 above) on the reduced data, obtaining a posterior sample of size 1000 for each data reduction size. The burn in values for the reduced samples were set to be the same as for the original unreduced data.

Our goal is now to compare the posterior samples that were obtained by running the Gibbs sampler on the reduced data to the posterior samples that were obtained from the original data. In order to do so, we use a total of three different measures.

The first measure of approximation quality that we apply compares the mean of the original posterior distribution without data reduction to the mean of the posterior distribution with a prior data reduction step, by evaluating the expression $\|\mu_\beta - \widetilde{\mu}_\beta\|_2$, where μ_β is the mean of the original posterior and $\widetilde{\mu}_\beta$ is the mean of the reduced posterior.

Likewise, our second measure compares the covariance matrices of the two samples by evaluating the expression $\|\Sigma_\beta - \widetilde{\Sigma}_\beta\|_2$, where Σ_β is the covariance matrix of the original posterior sample without data reduction and $\widetilde{\Sigma}_\beta$ is the covariance matrix of the posterior sample with data reduction. The norm $\|\cdot\|_2$ refers to the ℓ_2 matrix norm, which is equal to the largest singular value.

Our third measure is the maximum mean discrepancy (MMD, see for example [Gretton et al., 2007]), which was also applied in [Huggins et al., 2016]. The MMD was originally intended to be a test statistic for the problem of testing, if two samples come frome the same distribution or not. In order to do so, the idea behind the MMD is to transform both samples into a higher dimensional space and then to compare the means of the samples in the new space. The MMD is then the resulting difference of the means in the higher dimensional space. It follows, that the higher the MMD, the more different the two samples are. In order to transform the samples into the higher space, the authors of [Huggins et al., 2016] used a so called polynomial kernel function, and to stay comparable to them, we use a polynomial kernel as well.

5.3.3 Comparison of Approximation Quality

Figure 6 shows the results of our experiment for the measures $\|\mu_\beta - \widetilde{\mu}_\beta\|_2$, which we from now on call mean distance and $\|\Sigma_\beta - \widetilde{\Sigma}_\beta\|_2$, which we from now on call covariance distance. The results for the MMD can be found in Figure 7.

For the Covertype dataset, we can see that our two algorithms substantially outperform the uniform sampling algorithm for each of the three measures. The clearest difference can be seen when looking at the covariance distance. While both of our algorithms approach zero even for reduction sizes of less than 10000, the uniform sampling has big trouble to even come close to convergence. This finding suggests, that the samples of the posterior that were generated after running our algorithms are substantially

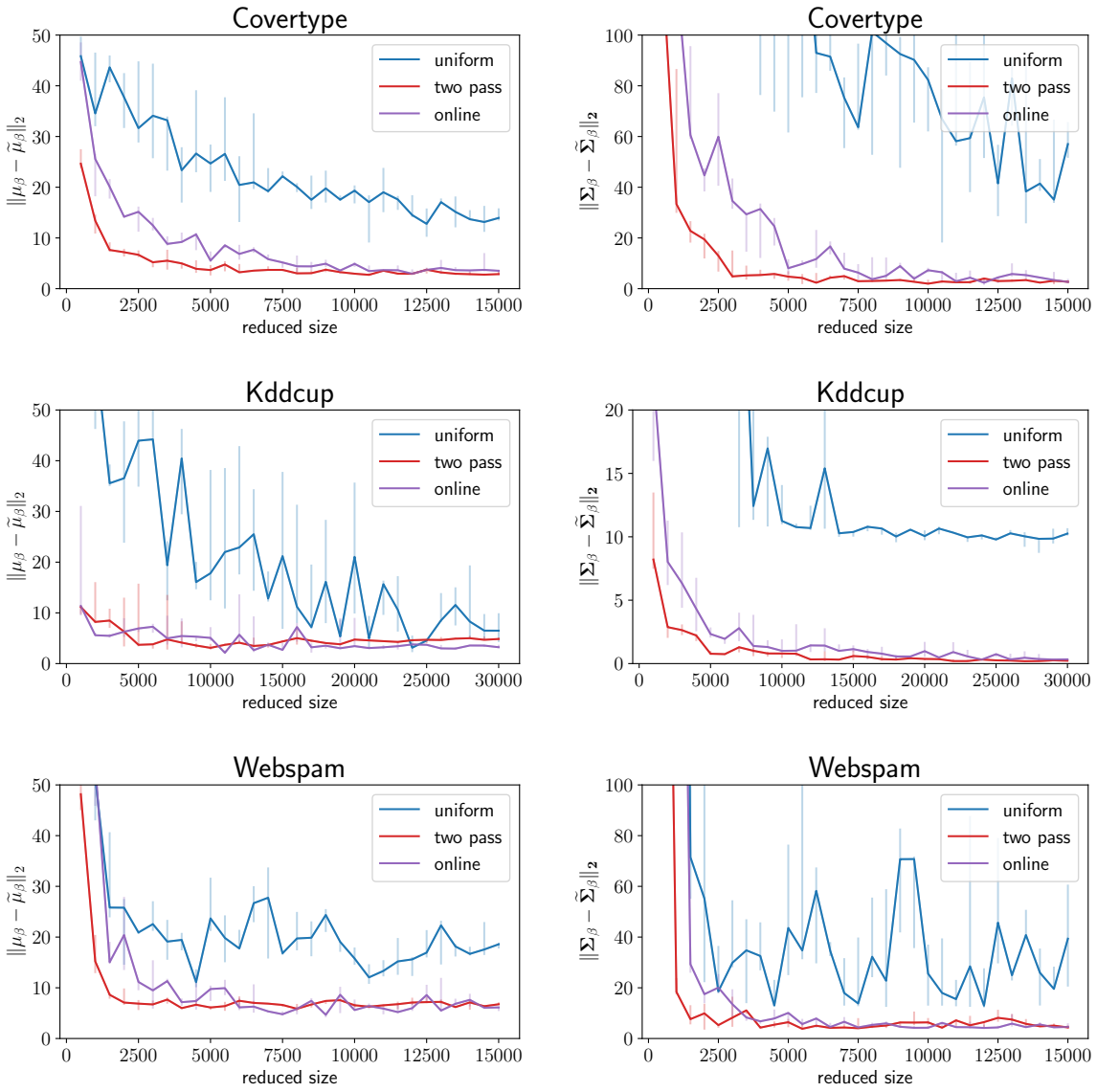

Figure 6: In the left column, we see the mean distance between the posterior distribution sampled after running our algorithms and the original posterior distribution sampled without data reduction. In the right column, we see the covariance distance for our algorithms. Every experiment was repeated a total of 5 times, and the solid lines represent the medians. The lower and upper error bars indicate the 25% quantile as well as the 75% quantile, respectively.

closer to the original posterior distribution in terms of variance, than the samples generated by using the uniform sampling procedure. When looking at the other measures for the Covertype dataset, we can make similar observations: Despite the fact that we characterized the Covertype dataset to be well behaved for uniform sampling earlier on, we now observe that the uniform sampling seems to fail to capture some of the important elements of the data, which substantially influence the outcome of the Gibbs sampling procedure.

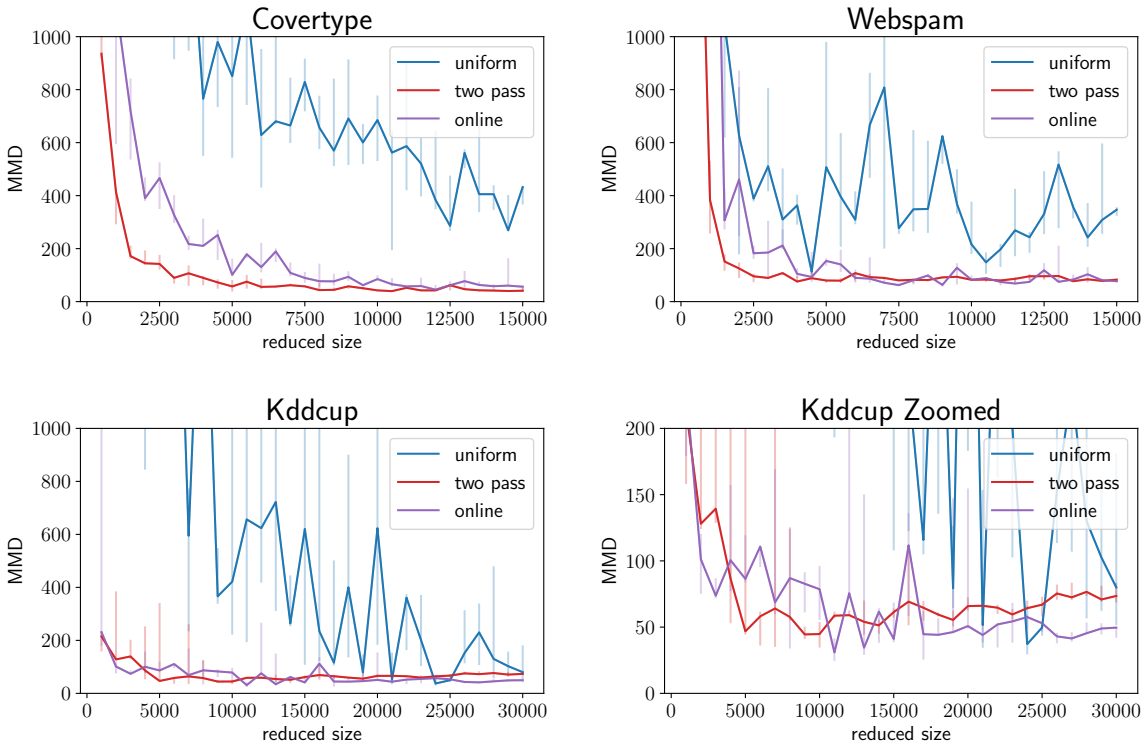

Figure 7: The MMD for each dataset and multiple data reduction sizes. Every experiment was repeated a total of 5 times, and the solid lines represent the medians. The lower and upper error bars indicate the 25% quantile as well as the 75% quantile, respectively.

For the Kddcup dataset, the differences are particularly apparent when looking at the covariance distance. Here, the uniform sampling even seems to stagnate, not moving towards zero at all. Our two algorithms on the other hand don't have the same problem: Their covariance distance decreases and approaches zero without any visible trouble. When looking at the mean distance for the Kddcup dataset, we can see that our algorithms also outperform, but at least the uniform sampling comes close for reduction sizes of over 25000. For the MMD, we can see a similar picture: Our algorithms clearly outperform, but the uniform sampling at least comes closer for the higher reduction sizes, although the error bars indicate a much higher degree of instability.

When looking at the Webspam dataset, we can see a similar picture like we already saw for the other two datasets. For each of the three measures, uniform sampling exhibits substantial problems, while our leverage score based algorithms seem to deliver much higher quality approximations.

To conclude the discussion on approximation quality, we compare a concrete sample of the posterior, that was created by using each of the three algorithms with a reduction size of 15000, to the original posterior sample of the Covertype dataset. In Figure 8 and Figure 9, we present boxplots to visualize the samples for each of the coefficients.

For the first part of the spectrum, β_0 to β_8, we can see that the posterior samples generated by each of the algorithms struggle to approximate the median as well as

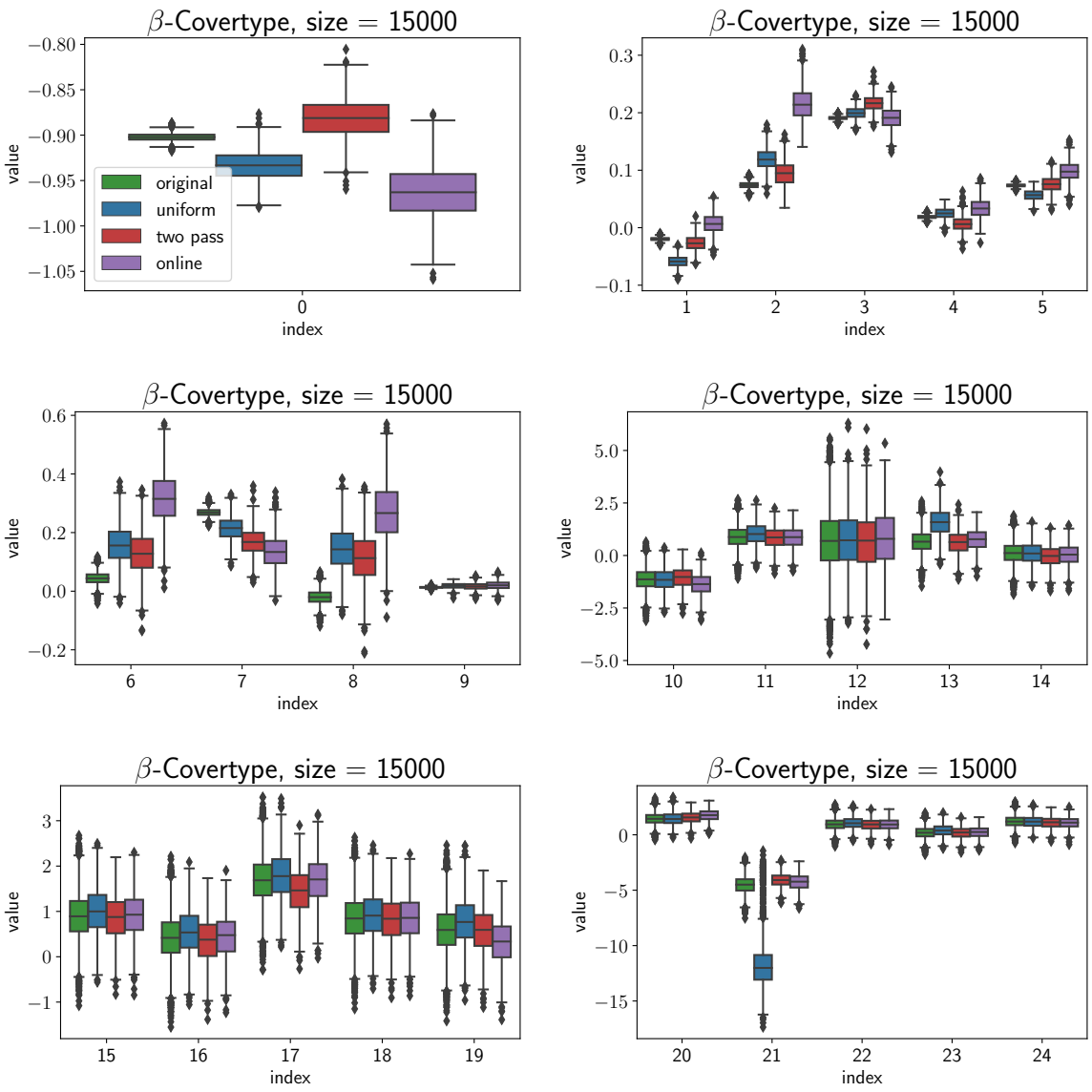

Figure 8: The posterior samples obtained after running each of the algorithms for a reduction size of 15000 are compared to the original sample of the posterior distribution of the Covertype dataset by using boxplots for each of the dimensions of β. Here, we can see β_0 to β_{24}, the remainder can be found in Figure 9.

the variance of the original posterior distribution. However, for the remainder of the coefficients, the approximation seems to be very close for our two algorithms as well as for the uniform sampling procedure, with the exception of β_{21}, β_{25}, β_{47} and β_{50}, where the uniform sampling procedure fails. Thus, we can see that for the majority of the spectrum, excluding the coefficient β_0 to β_8, our algorithms already seem to be delivering decent approximations that could potentially even be used insead of a full posterior sample.

It remains to be investigated though, why all the algorithms struggle with the first

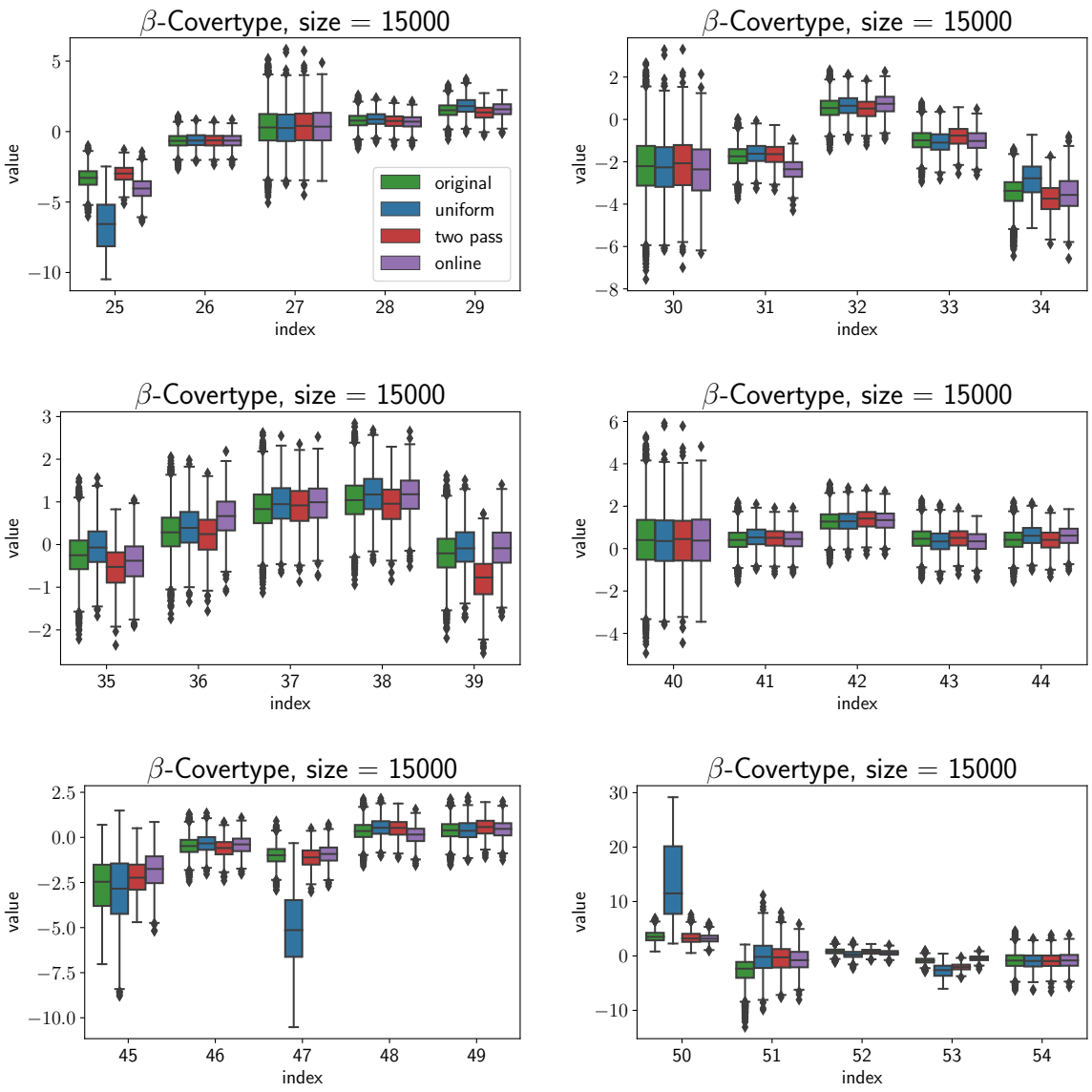

Figure 9: The boxplots comparing the remainder β_{25} to β_{54} of the posterior distributions. The intercept term is denoted by β_{54}.

coefficients β_0 to β_8. This could perhaps be due to an insufficient reduction size on the one hand or maybe even an insufficiently small burn in value on the other hand, but we leave this investigation as an open problem. The most important finding of this section is, that both of our algorithms outperform the baseline uniform sampling method by a substantial margin. To conclude our discussion of the experiments, we again compare the runtimes of our algorithms, but this time for the Bayesian setting.

5.3.4 Comparison of Running Times

To compare the running times of our algorithms, we take a look at the different sampling rates, i.e. the samples per second that the Gibbs sampler was able to generate with and without applying our algorithms first. The corresponding data can be found in Table 2, we list the sampling rates both including and excluding the time it takes for the algorithms to reduce the data before running the Gibbs sampler. We note, that for the same reasons as in Section 5.2.2, the online algorithm had to be excluded from the comparison.

Method	**Size**	**Dataset**	**samples / second (incl. red. time)**	**samples / second (excl. red. time)**
No reduction	581012	Covertype	1.5	1.5
two pass	15000	Covertype	36	38
uniform	15000	Covertype	46	46
two pass	1000	Covertype	116	153
uniform	1000	Covertype	176	176
No reduction	494021	KDDCup	4	4
two pass	15000	KDDCup	21	21
uniform	15000	KDDCup	107	107
two pass	1000	KDDCup	85	90
uniform	1000	KDDCup	619	619
No reduction	350000	Webspam	2	2
two pass	15000	Webspam	13	14
uniform	15000	Webspam	28	28
two pass	1000	Webspam	60	62
uniform	1000	Webspam	84	84

Table 2: The sampling rates of the Gibb sampler for generating the posterior samples with and without applying a data reduction method first. The sampling rates are given including and excluding the reduction time, i.e. the time it takes for the algorithms to reduce the size of the original dataset. We note, that for the same reasons as in Section 5.2.2, the online algorithm was excluded from the comparison.

When looking at the sampling rates for the Covertype dataset, we can see that both, the uniform sampling algorithm as well as our two pass algorithm, achieve considerable speed ups for the Gibbs sampler. While the Gibbs sampler can only generate roughly 1.5 samples every second for the full dataset, a prior reduction to 15000 samples leads to 36 samples per second for the uniform sampling algorithm and 46 samples per second for the two pass algorithm (including the reduction time), which is an increase by a factor of 30 and 24, respectively. For reduction sizes of only 1000, we can achieve even higher boosts of the sampling rate, but we note that the approximation quality might not be ideal for such low reduction sizes, as discussed in the previous section. We can also see, that for higher reduction sizes of 15000, the sampling rates including the reduction time are close to the sampling rates excluding the reduction time for the two pass algorithm,

which indicates that the time it takes to reduce the data amortizes itself the longer the Gibbs sampler runs.

It is also interesting to note, that the Gibbs sampler can generate samples slightly quicker, when presented with a subset of the original data that was obtained by uniform sampling, compared to a subset of the same size that was obtained by the two pass algorithm. A reason for this could perhaps be, that the leverage score based two pass algorithm tends to pick up on outliers more frequently, which could potentially slow down the sampling processes from the truncated normal distribution. In our implementation of the Gibbs sampler, the truncated normal samples are drawn via rejection sampling, if the probability of not rejecting a sample is sufficiently large ($\geq 5\%$), and otherwise it falls back to a slower implementation which works better for outliers. It could be possible, that when more outliers are present in the reduced sample, the slower implementation is triggered more frequently, which reduces the sampling rate.

Taking a look at the sampling rates for the Kddcup dataset, we can observe a similar pattern, but this time the differences between the two pass algorithm and the uniform sampling algorithm are greater. While the two pass algorithm for a reduction size of 15000 only achieves speed ups of a factor of roughly 5 (including the reduction time), the uniform sampling method allows the Gibbs sampler to increase the samples per second by a factor of more than 26. Here, we can see that the increased approximation quality of the two pass algorithm doesn't come without a price. Still, what good is the fast generation of posterior samples, if the quality is lacking? As we saw in the previous section, the two pass algorithm outperformed the uniform sampling method on the Kddcup dataset in each of our quality measures.

When comparing the sampling rates of uniform sampling to the two pass algorithm excluding the time it takes to reduce the data, we can see even greater differences for the Kddcup dataset. Here it seems, that the subsets obtained by the two pass algorithm are causing the Gibbs sampler to slow down way more than the subsets obtained by uniform sampling. This could potentially be explained by the fact that the Kddcup contains a lot more outliers that the leverage score based two pass algorithm can pick up on, which could slow down the sampling from the truncated normal distribution.

Taking a look at the Webspam dataset, we can see the same overall pattern that we already observed for the other two datasets. Both, the uniform sampling procedure as well as the two pass algorithm are able to increase the sampling rate of the Gibbs sampler for reduction sizes of 15000 by a factor of 6 for the two pass algorithm and 14 for uniform sampling, including the reduction time. However, when excluding the reduction times from the sampling rates, the differences between uniform sampling and the two pass algorithm aren't quite as substantial as for the Kddcup dataset, but still higher as for the Covertype dataset. This could, again, be due to the fact that the Webspam dataset contains a few heavy hitters that could potentially slow down the Gibbs sampler. On the other hand, the Webspam dataset was shown not to be as irregular as the Kddcup dataset, which would explain why the differences aren't that extreme.

To sum up the discussion of our experiments on coreset-based Bayesian inference, we successfully showed that both of our algorithms enable us to achieve a far greater approximation quality of the posterior distribution than the uniform sampling procedure.

This increased quality comes at a price of course, because both of our leverage score based algorithms are slower than the uniform sampling counterpart. It follows, that we are presented with a tradeoff: If we choose the uniform sampling algorithm for data reduction, we most likely get the highest increases in sampling rate for the Gibbs sampler, but the downside is that we suffer from instability issues and poor approximation quality. On the other hand, if we choose one of our coreset algorithms, we enjoy the provable guarantees of the sensitivity framework, which lead us to more reliable and stable approximations than the uniform sampling procedure, especially for not so well behaved datasets like Kddcup, while still gaining decent speed ups for the Gibbs sampler, albeit not as impressive as for the uniform sampling. Still, if the practitioner of a Bayesian probit analysis is faced with the decision which data reduction algorithm to choose, he or she will pick one of the leverage score based algorithms, if the challenge is to obtain stable approximations of guaranteeably high quality, while still enjoying some decent performance gains.

6 Contributions

We conclude this work by recapitulating the contributions we delivered during our analysis of coreset-based data reduction algorithms for the probit model.

As a first step, we were able to show, that not all dataset allow for obtaining small $(1 \pm \epsilon)$ approximations of the probit loss function, but we managed to overcome this obstacle by restricting our analysis to only those datasets, where a probit model can successfully be applied by using the method of maximum likelihood estimation. Further, we extended the notion of μ-complexity by [Munteanu et al., 2018] to the realm of probit analysis and showed, how μ-complexity is linked to the existence and uniqueness of the maximum likelihood estimator of the probit model, making it a necessary precondition for our subsequent advances in constructing the data reduction algorithms.

In the next step, we introduced the notion of coresets as well as the sensitivity framework and thereby outlined a roadmap, which we would follow towards our ambition of finding our first coreset construction algorithm. Adhering to the theory of the sensitivity framework, we first showed that sampling proportionally to the statistical leverage scores yields small sensitivity bounds and we also managed to control the VC dimension of our problem by applying the technique of leverage score rounding, thus arriving at our first provably correct coreset algorithm that can reduce μ-complex dataset to a size with a leading term in only $O(\mu d^2 \log(n))$.

Having successfully constructed our first data reduction algorithm, we quickly noticed that there were some substantial issues regarding its running time and efficiency, that still needed improvement. In order to do so, we applied the sketching methods outlined in [Drineas et al., 2012] and [Clarkson and Woodruff, 2017] to obtain our fast two pass coreset algorithm. In an attempt to adapt the fast two pass algorithm to situations, where two passes are not enough and each sampling decision has to be made immediately, we made use of the ideas in [Cohen et al., 2020] and [Chhaya et al., 2020] to obtain an online coreset algorithm, which only requires a single pass over the dataset.

To round off our work and demonstrate the practical applicability of our algorithms, we conducted a variety of experiments on three real world datasets, both in the domain of maximum likelihood estimation as well as in the Bayesian setting. Our experiments showed, that the algorithms we derived outperform the baseline uniform sampling algorithm with regards to approximation quality in nearly all situations by a significant margin. Only in terms of running time, the uniform sampling algorithm still reigns superior, although its gains in execution speed are overshadowed by its glaring instabilities and oftentimes mediocre results in approximation quality. Thus, for the data analysis practitioner who seeks reliable methods for reducing vast amounts of data with the goal of efficiently carrying out a probit analysis, our algorithms were demonstrated to be viable options, both in the frequentist domain of maximum likelihood estimation and in the Bayesian setting as well.

References

[Agresti, 2015] Agresti, A. (2015). *Foundations of Linear and Generalized Linear Models*. Wiley.

[Albert and Chib, 1993] Albert, J. H. and Chib, S. (1993). Bayesian analysis of binary and polychotomous response data. *Journal of the American Statistical Association*, 88(422):669–679.

[Braverman et al., 2016] Braverman, V., Feldman, D., and Lang, H. (2016). New frameworks for offline and streaming coreset constructions. *CoRR*, abs/1612.00889.

[Chao, 1982] Chao, M. T. (1982). A general purpose unequal probability sampling plan. *Biometrika*, 69(3):653–656.

[Chhaya et al., 2020] Chhaya, R., Choudhari, J., Dasgupta, A., and Shit, S. (2020). Streaming coresets for symmetric tensor factorization. *CoRR*, abs/2006.01225.

[Clarkson and Woodruff, 2017] Clarkson, K. L. and Woodruff, D. P. (2017). Low-rank approximation and regression in input sparsity time. *J. ACM*, 63(6).

[Cohen et al., 2020] Cohen, M. B., Musco, C., and Pachocki, J. (2020). Online row sampling. *Theory of Computing*, 16(15):1–25.

[Demidenko, 2001] Demidenko, E. (2001). Computational aspects of probit model. *Mathematical Communications*, 6:233–247.

[Drineas et al., 2012] Drineas, P., Magdon-Ismail, M., Mahoney, M. W., and Woodruff, D. P. (2012). Fast approximation of matrix coherence and statistical leverage. *Journal of Machine Learning Research*, 13(111):3475–3506.

[Fahrmeir et al., 2013] Fahrmeir, L., Kneib, T., Lang, S., and Marx, B. D. (2013). *Regression*. Springer-Verlag Berlin Heidelberg.

[Feldman and Langberg, 2011] Feldman, D. and Langberg, M. (2011). A unified framework for approximating and clustering data. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, page 569–578, New York, NY, USA. Association for Computing Machinery.

[Feldman et al., 2020] Feldman, D., Schmidt, M., and Sohler, C. (2020). Turning big data into tiny data: Constant-size coresets for k-means, pca, and projective clustering. *SIAM Journal on Computing*, 49(3):601–657.

[Gelfand and Smith, 1990] Gelfand, A. E. and Smith, A. F. M. (1990). Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85(410):398–409.

[Gelman et al., 2013] Gelman, A., Carlin, J., Stern, H., Dunson, D., Vehtari, A., and Rubin, D. (2013). *Bayesian Data Analysis*. Chapman and Hall/CRC, 3 edition.

[Geppert et al., 2017] Geppert, L. N., Ickstadt, K., Munteanu, A., Quedenfeld, J., and Sohler, C. (2017). Random projections for bayesian regression. *Stat. Comput.*, 27(1):79–101.

[Golub and van Loan, 2013] Golub, G. H. and van Loan, C. F. (2013). *Matrix Computations*. JHU Press, fourth edition.

[Gordon, 1941] Gordon, R. D. (1941). Values of mills' ratio of area to bounding ordinate and of the normal probability integral for large values of the argument. *The Annals of Mathematical Statistics*, 12(3):364–366.

[Gretton et al., 2007] Gretton, A., Borgwardt, K., Rasch, M., Schölkopf, B., and Smola, A. (2007). A kernel method for the two-sample-problem. In Schölkopf, B., Platt, J., and Hoffman, T., editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press.

[Haberman, 1974] Haberman, S. J. (1974). *The Analysis of Frequency Data*. The University of Chicago Press.

[Huggins et al., 2016] Huggins, J. H., Campbell, T., and Broderick, T. (2016). Coresets for scalable bayesian logistic regression. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 4087–4095, Red Hook, NY, USA. Curran Associates Inc.

[Johnson and Lindenstrauss, 1984] Johnson, W. B. and Lindenstrauss, J. (1984). Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics*, 26(1):189–206.

[Jolliffe, 2002] Jolliffe, I. (2002). *Principal Component Analysis*. Springer Series in Statistics. Springer-Verlag New York, 2 edition.

[Kearns and Vazirani, 1994] Kearns, M. J. and Vazirani, U. V. (1994). *An Introduction to Computational Learning Theory*. MIT Press.

[Kremer et al., 1999] Kremer, I., Nisan, N., and Ron, D. (1999). On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49.

[Langberg and Schulman, 2010] Langberg, M. and Schulman, L. J. (2010). Universal ϵ-approximators for integrals. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 598–607.

[Lesaffre and Kaufmann, 1992] Lesaffre, E. and Kaufmann, H. (1992). Existence and uniqueness of the maximum likelihood estimator for a multivariate probit model. *Journal of the American Statistical Association*, 87:805–811.

[Mai et al., 2021] Mai, T., Rao, A. B., and Musco, C. (2021). Coresets for classification - simplified and strengthened. *CoRR*, abs/2106.04254.

[McCullagh and Nelder, 1989] McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*. Chapman and Hall/CRC.

[Munteanu and Schwiegelshohn, 2018] Munteanu, A. and Schwiegelshohn, C. (2018). Coresets-methods and history: A theoreticians design pattern for approximation and streaming algorithms. *KI - Künstliche Intelligenz*, 32(1):37–53.

[Munteanu et al., 2018] Munteanu, A., Schwiegelshohn, C., Sohler, C., and Woodruff, D. P. (2018). On coresets for logistic regression. In *Advances in Neural Information Processing Systems 31, (NeurIPS)*, pages 6562–6571.

[Nocedal and Wright, 2006] Nocedal, J. and Wright, S. (2006). *Numerical Optimization*. Springer-Verlag New York, 2 edition.

[Sherman and Morrison, 1950] Sherman, J. and Morrison, W. J. (1950). Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127.

[Tanner and Wong, 1987] Tanner, M. A. and Wong, W. H. (1987). The calculation of posterior distributions by data augmentation. *Journal of the American Statistical Association*, 82(398):528–540.

[Vaart, 1998] Vaart, A. W. v. d. (1998). *Asymptotic Statistics*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press.

[Wedderburn, 1976] Wedderburn, R. W. M. (1976). On the existence and uniqueness of the maximum likelihood estimates for certain generalized linear models. *Biometrika*, 63(1):27–32.

Eidesstattliche Versicherung

(Affidavit)

Peters, Christian

Name, Vorname
(surname, first name)

213996

Matrikelnummer
(student ID number)

☐ Bachelorarbeit
(Bachelor's thesis)

■ Masterarbeit
(Master's thesis)

Titel
(Title)

Data Reduction for Efficient Probit Regression

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present thesis with the above-mentioned title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution before.

Ort, Datum
(place, date)

Unterschrift
(signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to EUR 50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the Chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, Section 63 (5) North Rhine-Westphalia Higher Education Act (*Hochschulgesetz, HG*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund University will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:*

Ort, Datum
(place, date)

Unterschrift
(signature)

***Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.**