Sampling Bias Correction for Supervised Machine Learning
A Probabilistic, Bayesian Approach

Max Sklar

October 28, 2021

Abstract

Given a supervised machine learning problem where the dataset has been subject to sampling bias, we ask if we can still select or search potential models that best fit the original dataset if the sampling method is known. While information is of course lost with data removal, a formula can be derived to best alter the posterior probabilities in the model space in order to do just that. We derive that formula. We start by setting up a standard Bayesian framework for supervised machine learning, followed by the introduction of a sampling function along with an argument for how this affects the posterior distribution. We also discuss scenarios where a dataset might be intentionally sampled unevenly, including the case of rare events. The general bias correction formula is then applied to the common case of binary logistic regression. We finally discuss several different sampling methods, including those that aren't covered by this framework but offer opportunities for generalization.

1 Introduction

A basic concept in computer science is a *function* $f : X \to Y$, which is an *algorithm* or a series of precise instructions for computing a value in set Y from an input in set X. The goal of *supervised machine learning* is to develop, execute, and deploy algorithms which learn to compute such functions through example rather than direct instructions. This type of machine learning algorithm learns to approximate the function $f : X \to Y$ given a dataset consisting of *instances* of input-output pairs $(x, y) \in X \times Y$. This set of examples, the *training set*, is what allows the machine to learn the *concept* of f.

The composition of the training set is crucial, and it s expected that the model produced will be evaluated and ultimately deployed on data that similar to the training set, as if these *instances* were all pulled from the same distribution. Sometimes models can perform well on datasets with somewhat different compositions, and in practice we see this relied upon from time to time. However, that assumption is far from certain to hold.

In many practical examples, the full dataset of instances is not available either by design or circumstance, and all that remains is biased sample. **Fortunately, when the sampling function is known, there is a method to learn the concept f from the remaining data, and to use all the evidence from that data as best as possible to learn a model to fit the *original* dataset.**

The mathematical formula becomes apparent when analyzing it through the *Bayesian framework* for supervised machine learning, reviewed in section 3. Here, we start with a probability distribution over large set of

possible models. Each model provides for any input $x \in X$ not a singular output $y \in Y$ but a probability distribution over possible outputs in Y. We are then able to use the dataset to compute a *posterior distribution* over the potential models, allowing us to discriminate between them by their projected performance.

The posterior distribution in the case of an unevenly sampled dataset is found mathematically in sections 4 and 5. The tractability of the final step to select or sample a model depends on the specific problem and approach to dealing with the posterior distribution. In most cases tractability of searching the solution space is not compomised. In section 6 we apply this formula to binary logistic regression as an particularly common example.

2 Motivation

While big data has unlocked incredible new applications of machine learning, the accompanied processing comes at a heavy cost in terms of time, money, and energy. Many algorithms will perform just as well had they been trained on a fraction of the data available, or at least the performance difference is not enough to justify the cost of including all data. Therefore, sampling or removing data from the training set is a legitimate design decision.

One way to slim down data is through uniform random sampling, where each instance in the original dataset has an equal probability of inclusion. This process ignores the fact that some training examples are more valuable than others, particularly when there is a classification imbalance.

2.1 Example: Theoretical Image Recognition

Suppose that the goal was to train an image recognition algorithm to recognize the image of a lion. Furthermore, the training data contains 2000 images of lions, and about 400,000 images that are **not** lions. The following 3 schemes for sampling are proposed:

(A) Train on all of the data (2000 Lions, 400000 Non-Lions)

(B) Randomly sample 25 percent of the data (500 Lions, 100000 Non-Lions)

(C) randomly sample a quarter of ONLY the non-lion images (2000 Lions, 100000 Non-Lions)

In order to make a decision, one would now ask about the likely outcome or tradoffs between these 3 options.

Option A uses all of the data, and takes the most resources to train. In theory, the model produced by option A will be better than (or at least as good as) any of the other models. This is because in the Bayesian framework we search for a model that best explains the data, and additional data can only refine our search. In practice this might not be the case because the algorithm for searching the model space is imperfect. For example, if a *mini-batch* system is used to search for the best model, it might not converge as quickly or as perfectly with an imbalanced dataset.

Option B has all the imbalance of A, but now with fewer instances. The algoithm will run more quickly with B, but the results are likely to be far worse. From an intuitive standpoint, an algorithms designed to learn what a lion is relies on lion photos, and for that we don't have very many. Because of the imbalance,

each lion photo is much more valuable to our model's success than a non-lion photo. Common sense would tell us that a reduction in lion photos from 2000 to 500 is likely to make a significant negative impact.

Under option C, the data size is almost as small as B, but now all 2000 lion photos remain. The likely outcome is that option C performs far better than option B. There's even a chance that it performs better than option A if the data imbalance hinders the algorithm, but now the resources to run it will be low as in option B.

One concern is that if option C doesn't correct for this bias and take the sampling into account, it will use learn that Lions are more common than they really are in the underlying dataset, and therefore will tend to overpredict lions. It is possible that the underlying image recognition algorithm will detect the core visual features of a lion and will not have that issue even without correction, but such a correction is desirable.

2.2 Example: Probabilistic Event Detection

While bias correction in the image recognition example is desirable, in some cases it is crucial! One example is the case of a probabilistic model where the results are expected to be highly uncertain and rely on accurate probabilities of the potential outputs.

One example of this is an iteration of Foursquare's attribution model, which is patent pending[11]. As part of a product that measures the ability of an online advertisement to drive consumers to the physical location of a given retail chain, the Foursquare data pipeline first builds a model that calculates the base probability that any given individual would visit that chain on any given day.

Foursquare's data set has many examples of visits, but the examples of people who "did not visit" on any given day is higher by several orders of magnitude. Therefore, it makes sense to downsample those non-visits. Because a precise probability is necessary for a high-end ad measurement product, that sampling must be accounted for appropriately and precisely.

2.3 Similar and Adjacent Work

The literature on sampling and rare events is quite vast, and no comprehensive review will be given here. Instead we refer to some selected work in this field which either inspired or was helpful in the course of this research.

For general cataloging of situations with imbalanced data and algorithmic techniques for it's mitigation, see the work of Maalouf and Trafalis[7]. The formulas for readjusting model estimations in the case of logistic regression can be found in Maalouf and Siddiqi[6]. For an in-depth discussion on rare events in logistic regression, the problems associated with it, and the mathematics of parameter estimation, see King and Zeng[4].

3 Bayesian Supervised Machine Learning Framework

The following is a typical setup for probabilistic supervised machine learning using a Bayesian framework, a variation of what can be found in general treatments of probabilistic inference[8][2][1]. The terminology and

variable names will be reused in subsequent sections.

Let X represent the input space. Let Y represent the output space.

The goal is to predict an output $y \in Y$ from a corresponding input $x \in X$.

The dataset used for training D consists of N examples of pairs $(x, y) \in (X \times Y)$. This dataset may be generated by an oracle from which we can ask for an arbitrary number of examples, or it might be a small and limited collection. In any case, our learning will be based off of just these N examples.

For all $n \in (0, 1, 2, \ldots, N-1)$, label the specific instance (x_n, y_n).

The strategy for attacking this problem is to create a *hypothesis space* H whose members $h \in H$ each encode a potential solution to the prediction problem. We assume that one of these predictors is the correct one, but they are all possible.[1] Therefore, each $h \in H$ can be considered a *hypothesis* for a solution to our problem. They are also called *predictors* or *models*. We will use the term predictor to empasize it's ultimate purpose while using the letter h.

In some setups, each hypothesis $h \in H$ is a direct function from X to Y, which means that these predictors are *directly predicting* $y \in Y$. Here we assume that each hypothesis is a function of $x \in X$ and returns a *probability measure* over Y. This is what we mean by *probabilistically predicting* $y \in Y$.

While the equations here generalize to all probability measures over the output space Y, we keep several cases in mind.

1. Y is finite. This is a problem of *classification*. Each predictor takes an input $x \in X$ and return a probability for each $y \in Y$ that sums to one.

2. Y is discrete but infinite. The predictor still assigns a number to each potential output, but now the infinite sum adds to one.

3. Y is continuous. The predictor returns a *probability distribution function* (PDF) over Y, which integrates to one. This predictor could put a positive probabiliy on specific values as well, making it a discrete-continuous hybrid.

In all of these cases, the model assigns a number to each $y \in Y$ which can be used to compute the relative probability of each potential output given an input x. In the case of discrete Y this relative probability of $y_1, y_2 \in Y$ is just the ratio of their individual probabilities, and for continuous Y it is the ratio of their values in the PDF.

Given this setup, we can say that each predictor h is uniquely identified with an (unnormalized) *relative probability function* $f_h : X \times Y \to \mathbb{R}^+$. Our generalization includes the possibility of an *improper probability distribution function*, for example one that assigns a score of 1 to each real number $Y = \mathbb{R}$ even though such a PDF cannot be normalized.

In cases where this probability function can be normalized, we use a normalized probability function $\hat{f}_h : X \times Y \to \mathbb{R}^+$.

[1] In practice we must consider that this space may not contain our solution, but this temporary assumption makes Bayesian inference possible.

$$\hat{f}_h(x,y)\left(\sum_{y'\in Y} f_h(x,y')\right) = f_h(x,y) \qquad \hat{f}_h(x,y)\left(\int_{y'\in Y} f_h(x,y')\right) = f_h(x,y). \tag{1}$$

The *prior distribution* over H is a probability measure representing the current belief over which predictor is correct. Typically, this will involve defining an *uninformative prior* which encodes absense of knowledge of the problem. It is also common to incorporate *Occam's Razor* which penalizes more complex predictors, or to use a prior that will be mathematically convenient. We will use $\mathbf{P}(h)$ as the *prior probability* of predictor $h \in H$.

The *posterior distribution* over H is a probability measure representing the belief over which predictor is correct **after** the data has been taken into account.

Once H has been defined along with a prior and a posterior, the task is either to *select* or *sample* a predictor that best explains the data. This process is often described as the learning algorithm itself because it is such a central part of it.

In the case of selection, the goal is to find a single predictor that best fits the data. A good example is the *maximum a posteriori* (MAP) estimate which seeks to find the most likely predictor. Another example is the *maximum likelihood estimate* (MLE) which fails to take the prior into account and just finds the predictor that assigns the highest probability to the dataset as a whole.

In sampling, a series of predictors are randomly generated according to their relative strength in explaining the data. Sampling is a way to approximate pulling a hypothesis randomly from its posterior distribution. We again rely on the concept of *relative probability*, this time between two potential predictors $h_1, h_2 \in H$. In general, the hypothesis space H is very complex and a variety of methods for selection and sampling have been developed.[10][3]

Bayes rule for discrete H finds the probability of individual predictor h_i, and in the continuous case it allows us to produce PDF values. $\mathbf{P}(D|h)$ is the probability of seeing the outputs in dataset D under the predictor h.

$$\mathbf{P}(h|D) = \frac{\mathbf{P}(D|h)\mathbf{P}(h)}{\sum_{h\in H}\mathbf{P}(D|h)} \qquad \mathbf{P}(h|D) = \frac{\mathbf{P}(D|h)\mathbf{P}(h)}{\int_{h\in H}\mathbf{P}(D|h)} \tag{2}$$

This is rewritten for relative probability to derive unnormalized values for $\mathbf{P}(h|D)$.

$$\mathbf{P}(h|D) \propto \mathbf{P}(D|h)\mathbf{P}(h) \tag{3}$$

The relative probability of the dataset under model h, denoted by $\mathbf{P}(D|h)$, is the product of the relative probability function of each instance. This allows us to rewrite the formula for the relative probability of a predictor as

$$\mathbf{P}(h|D) \propto \left(\prod_{n\in N} f_h(x_n, y_n)\right)\mathbf{P}(h).$$

It is often more convenient to work with a *negative log-likilihood loss* function[2] by applying $-\ln(\ldots)$ to the right hand side of this equation and getting

$$L(h) = -\sum_{n \in N} \ln(f_h(x_n, y_n)) - \ln(\mathbf{P}(h)).$$

Assume each hypothesis comes with its own negative log-likelihood loss function l_h where $f_h(x_n, y_n) \propto e^{-l_h(x_n, y_n)}$ and the prior $\mathbf{P}(h)$ can be reduced to a regularization function $\mathbf{r}(h)$ where $\mathbf{P}(h) \propto e^{-\mathbf{r}(h)}$. The final form of the loss function is

$$L(h) = \sum_{n \in N} l_h(x_n, y_n) + \mathbf{r}(h).$$

This form is convenient for model selection and sampling techniques such as hill climbing, gradient descent, or Markov Chain Monte Carlo. A good example is the *No U-Turn Sampler*[3] popular in the PyMC3[9] probabilistic programming package for python.

4 The Sampling Problem

Consider the case where dataset D has been derived by downsampling a larger dataset D^+. Formally, we say that D was generated from D^+ with a sampling probability function $\mathbf{s}$.

We limit ourselves to samplers that consider each datapoint $(x, y) \in D^+$ independently. Therefore, $\mathbf{s} : X \times Y \to [0, 1]$.

This type of sampling is optimal for parallel computation because the sampler has no state other than it's inputs (x_i, y_i). It does exclude some common sampling types, some of which are covered in Section 7.

The key point of this paper is that when the sampling function is known, a probability distribution and loss function can still be computed from D for predictors that are designed to work on D^+.

5 The General Solution

Start with the unnormalized Bayes rule in equation (3), but now consider that the posterior distribution and likelihood both depend on the sampling function $\mathbf{s}$.

$$\mathbf{P}(h|D, \mathbf{s}) \propto \mathbf{P}(D|h, \mathbf{s})\mathbf{P}(h)$$

As before, calculate the likelihood as

[2]For a great treatment on loss functions and their various tradeoffs, see A Tutorial on Energy Based Learning by Lecun[5]

$$\mathbf{P}(D|h) = \prod_{n \in N} \mathbf{P}(y_n|x_n, h, \mathbf{s}).$$

Note that the expression $\mathbf{P}(y_n|x_n, h, \mathbf{s})$ conditions on x_n, model h and sampler $\mathbf{s}$. The question is the probability of producing the label y_n under these circumstances. The following *generative description* is a useful tool in understanding how the sampling function factors into the generation of these labels y_n.

1. Consider as given a specific input x_n, a predictor $h \in H$, and the sampling function $\mathbf{s}$. The predictor h encodes a probability distribution over Y through the relative probability function $f_h(x_n, y_n)$.
2. Sample from that probability distribution, and make this a candidate for y_n, called y_n^*
3. Compute the sampling rate $\mathbf{s}(x_n, y_n^*)$ and use that rate to probabilistically determine whether y_n^* is accepted.
   (a) If it is accepted, return $y_n = y_n^*$.
   (b) If it is not accepted, return to step 2 to generated another candidate.

The sampling function must eventually halt which means that it cannot be 0 for every potential candidate of any given x_n. This is a fair assumption because otherwise x_n would never appear in D to begin with.

We now use the generative description to produce a recursive equation for $\mathbf{P}(y_n|x_n, h, \mathbf{s})$.

Suppose we want to find the probability of selecting y_n and the first candidate is y. If y is accepted, this probability is equal to 1 if $y_n = y$ and 0 otherwise, given by the indicator function $[y_n = y]$. If it is not accepted, then we the probability reverts to the original value of $\mathbf{P}(y_n|x_n, h, \mathbf{s})$. Therefore, the probabilty of ultimately accepting y_n comes to

$$P(y_n|x_n, h, \mathbf{s}) = \mathbf{s}(x_n, y)\,[y_n = y] + (1 - \mathbf{s}(x_n, y))P(y_n|x_n, h, \mathbf{s}).$$

If Y is discrete, the normalized probability function of model h in equation (1) $\hat{f}$ can be used to sum over all possible candidates y and setup the recursive equation

$$\mathbf{P}(y_n|x_n, h, \mathbf{s}) = \sum_{y \in Y} \hat{f}_h(x_n, y)\big(\mathbf{s}(x_n, y)\,[y_n = y] + (1 - \mathbf{s}(x_n, y))P(y_n|x_n, h, \mathbf{s})\big). \tag{4}$$

With algebraic manipuation documented in appendix A, we solve for $\mathbf{P}(y_n|x_n, h, \mathbf{s})$, reduce $\hat{f}$ to f, and derive the formula

$$\mathbf{P}(y_n|x_n, h, \mathbf{s}, z_n) = \frac{f_h(x_n, y_n)\mathbf{s}(s_n, y_n)}{\sum_{y \in Y} f_h(x_n, y)\mathbf{s}(x_n, y)}. \tag{5}$$

For a continuous output space Y, derive the analogous equation

$$\mathbf{P}(y_n|x_n, h, \mathbf{s}) = \frac{f_h(x_n, y_n)\mathbf{s}(s_n, y_n)}{\int_{y\in Y} f_h(x_n, y)\mathbf{s}(x_n, y)}. \tag{6}$$

The feasibility of computing the sum or integral term depends on the structure of Y, but is at least easy when Y is finite and small[3]. The formula generalizes to more exotic probability measures on Y, though the rigorous treatment is outside our scope.

5.1 Formula for Negative Log-Likelihood

Equation (5) provides all the tools needed to assign relative probabilities to predictors in H. To get it in a more helpful form, we derive a negative log likelihood loss function starting with the relative Bayes formula.

$$\mathbf{P}(h|D, \mathbf{s}) \propto \prod_{n\in N} \left[\mathbf{P}(y_n|x_n, h, \mathbf{s})\right] \mathbf{P}(h)$$

Use equation (5) to get this in terms of f_h:

$$\mathbf{P}(h|D, \mathbf{s}) \propto \prod_{n\in N} \left[f_h(x_n, y_n)\mathbf{s}(s_n, y_n) \left[\sum_{y\in Y} f_h(x_n, y)\mathbf{s}(x_n, y)\right]^{-1}\right] \mathbf{P}(h)$$

Finally, derive a negative log likelihood loss function on h:

$$L(h) = \sum_{n\in N} \left[l_h(x_n, y_n) - \ln \mathbf{s}(s_n, y_n) + \ln \sum_{y\in Y} f_h(x_n, y)\mathbf{s}(x_n, y)\right] + \mathbf{r}(h)$$

6 Solution for Binary Logistic Regression

A *binary logistic regression* is a specific case of the supervised learning problem in section 3 with the following additional properties:

1. It is a *binary classification* in that $Y = \{0, 1\}$.
2. The input space X is a list of real valued features. Let F denotes a finite set of features, $X = \mathbb{R}^{|F|}$.
3. Each predictor $h \in H$ is parameterized by $h = (c, \mathbf{w})$ where $c \in \mathbb{R}$ is the intercept and $\mathbf{w} \in \mathbb{R}^{|F|}$ is an $|F|$-dimentional vector of weights corresponding to each input feature.

[3] "small" meaning small enough to enumerated by a machine or the machine that is doing the computation

Each hypothesis $(c, \mathbf{w}) = h \in H$ corresponds to the following probability distribution function:

$$f_{c,\mathbf{w}}(x_n, y_n) = \frac{e^{y_n \cdot (c + \mathbf{w} \cdot x_n)}}{1 + e^{c + \mathbf{w} \cdot x_n}}$$

Using equation (5) we see

$$\mathbf{P}(y_n | x_n, h, \mathbf{s}) = \frac{f_h(x_n, y_n)\mathbf{s}(s_n, y_n)}{\sum_{y \in Y} f_h(x_n, y)\mathbf{s}(x_n, y)} = \frac{e^{y_n \cdot (c + \mathbf{w} \cdot x_n)}\mathbf{s}(s_n, y_n)}{\mathbf{s}(x_n, 0) + e^{c + \mathbf{w} \cdot x_n}\mathbf{s}(x_n, 1)}$$

.

These problems are often framed to focus on the probability of the *target condition* $y_n = 1$. This target condition is usually the rare event that triggered the decision to use adaptive sampling in the first place. We can solve for this probability as follows:

$$\mathbf{P}(y_n = 1 | x_n, h, \mathbf{s}) = \frac{e^{c + \mathbf{w} \cdot x_n}\mathbf{s}(s_n, 1)}{\mathbf{s}(x_n, 0) + e^{c + \mathbf{w} \cdot x_n}\mathbf{s}(x_n, 1)} = \frac{e^{c + \mathbf{w} \cdot x_n}}{r_n + e^{c + \mathbf{w} \cdot x_n}}$$

Here, $r_n = \mathbf{s}(x_n, 0)(\mathbf{s}(x_n, 1))^{-1}$ is the true-to-false sample ratio for each instance n. This ratio is all that is needed to compute results for the binary logistic regression case. Note that for $r_n = 1$, we are left with the original binary logistic regression formula.

For the prior, we can use a Gaussian distribution (aka L2, or ridge regression) with weight λ so:

$$\mathbf{r}(c, \mathbf{w}) = \frac{1}{2}\lambda(\mathbf{w} \cdot \mathbf{w})$$

Put this together and drop some constant factors to derive a negative log-likelihood loss function.

$$L(h) = \sum_{n \in N} \left(\ln \left(r_n + e^{c + \mathbf{w} \cdot x_n} \right) - y_n \cdot (c + \mathbf{w} \cdot x_n) \right) + \frac{1}{2}\lambda(\mathbf{w} \cdot \mathbf{w})$$

The derivative on a single weight $\mathbf{w}_f$ can be computed into the following simple form in order to perform gradient descent.

$$\frac{\partial}{\partial \mathbf{w}_f} L(h) = \sum_{n \in N} \left(x_{n,f} \frac{e^{c + \mathbf{w} \cdot x_n}}{r_n + e^{c + \mathbf{w} \cdot x_n}} - y_n \cdot x_{n,f} \right) + \lambda \cdot w_f$$

$$\frac{\partial}{\partial \mathbf{w}_f} L(h) = \sum_{n \in N} x_{n,f} \left(\mathbf{P}(y_n = 1 | x_n, h, \mathbf{s}) - y_n \right) + \lambda \cdot w_f$$

7 Future Work

The fact that machine learning can be done effectively after sampling when using the Bayesian framework opens up several possibilities for improvements in statistical computation.

7.1 Simple Random Sampling

Simple Random Sampling is where an exact number of datapoints is retained in the dataset. The point-wise sampling function can never guarantee a given number of points will be selected. Simple Random Sampling can furthermore be stratefied by output to account for rare events.

7.2 Oversampling

The sampling function $\mathbf{s}$ assumes that each datapoint is either included in the dataset or excluded. This is known as *undersampling*. We could allow for *oversampling*, where some instances are included in D multiple times.

Now instead of $\mathbf{s} : X \times Y \to [0, 1]$, the sampling function s returns a probability distribution over all natural numbers. This could be any probability distribution, but in practice the *poisson distribution* is often used. The poisson distribution with parameter λ places probability of multiplicity k at $\frac{\lambda^k e^{-\lambda}}{k!}$.

Bootstrap sampling methods rely on oversampling.

7.3 Using General History in the Sampling Function

In our setup, the domain of s is $X \times Y$. It is possible for the sampling function to have a state where it can take into account the entire history of what has happened up to that point. For each past instance, we know the input X, the label Y, the probability assigned to it $p \in [0, 1]$, and the final decision $s \in 2 = 0, 1$.

Allowing for history can take into account more complex samplers.

7.4 The Value of an Instance

Finally, there is much work to be done in quantifying the expected value that an instance will bring in terms of including it in D. The complication here is that we don't know how much a datapoint will change the posterior distribution of H before it is included, and the goal of the original problem could have different values on the tradeoffs. For example, maybe the ability to classify certain inputs is more valuable than others.

Therefore, the ability to quickly evaluate the value that an instance provides versus the cost of including it would eliminate much unneccary work.

Additionally, perhaps seceral predictors can be created in a similar manner, each with a different sampling

function representing different valuations of the data points. If these are be combined in an *ensemble model*, they will likely perform better than in the singular approach. Because they can be run in parallel, this provides real engineering benefits.

Appendices

A Solving for the General Formula

Here we start with equation (4) and go through the series of steps neccesary to derive its final form in equation (5). Equation (4) beings as

$$\mathbf{P}(y_n|x_n,h,\mathbf{s}) = \sum_{y\in Y} \hat{f}_h(x_n,y)\big(\mathbf{s}(x_n,y)\,[y_n=y] + (1-\mathbf{s}(x_n,y))P(y_n|x_n,h,\mathbf{s})\big)$$

.

Break out the summation to get

$$\mathbf{P}(y_n|x_n,h,\mathbf{s}) = \sum_{y\in Y} \hat{f}_h(x_n,y)\mathbf{s}(x_n,y)\,[y_n=y] + \sum_{y\in Y} \hat{f}_h(x_n,y)(1-\mathbf{s}(s_n,y))P(y_n|x_n,h,\mathbf{s})$$

.

Note that in the first summation, the only non-zero addend is when $y = y_n$ and the corresponding indicator function is 1. Therefore, we can replace y with y_n and remove the summation. In the second sum, factor out $P(y_n|x_n,h,\mathbf{s})$ which does not contain summation index y.

$$\mathbf{P}(y_n|x_n,h,\mathbf{s}) = \hat{f}_h(x_n,y_n)\mathbf{s}(x_n,y_n) + P(y_n|x_n,h,\mathbf{s})\sum_{y\in Y} \hat{f}_h(x_n,y)(1-\mathbf{s}(s_n,y))$$

Collect the term $\mathbf{P}(y_n|x_n,h,\mathbf{s})$ and break down the remaining summation.

$$\mathbf{P}(y_n|x_n,h,\mathbf{s})\left[1-\sum_{y\in Y} \hat{f}_h(x_n,y)(1-\mathbf{s}(x_n,y))\right] = \hat{f}_h(x_n,y_n)\mathbf{s}(s_n,y_n)$$

$$\mathbf{P}(y_n|x_n,h,\mathbf{s})\left[1-\sum_{y\in Y} \hat{f}_h(x_n,y)+\sum_{y\in Y} \hat{f}_h(x_n,y)\mathbf{s}(x_n,y)\right] = \hat{f}_h(x_n,y_n)\mathbf{s}(s_n,y_n)$$

Because we are using the normalized probability function $\hat{f}$, and from equation (1) we know that for all possible inputs x, $\sum_{y\in Y} \hat{f}_h(x,y) = 1$, we can replace the summation over $\hat{f}$ and cancel as follows:

$$\mathbf{P}(y_n|x_n,h,\mathbf{s})\left[1-1+\sum_{y\in Y}\hat{f}_h(x_n,y)\mathbf{s}(s_n,y)\right]=\hat{f}_h(x_n,y_n)\mathbf{s}(x_n,y_n)$$

$$\mathbf{P}(y_n|x_n,h,\mathbf{s})\left[\sum_{y\in Y}\hat{f}_h(x_n,y)\mathbf{s}(s_n,y)\right]=\hat{f}_h(x_n,y_n)\mathbf{s}(x_n,y_n)$$

Finally, we note that because of the equation (1), the $\hat{f}_h$ terms are simply a constant factor of same terms with the unnormalized f_h. Because this normalization factor appears on both sides of the equation, $\hat{f}_h$ can be reduced to f through cancellation[4]. Thus with one extra step of division, the final form can be given as (5).

$$\mathbf{P}(y_n|x_n,h,\mathbf{s})=\frac{f_h(x_n,y_n)\mathbf{s}(s_n,y_n)}{\sum_{y\in Y}f_h(x_n,y)\mathbf{s}(x_n,y)} \tag{5}$$

If we assume that there was no sampling, or the sampling was uniform where $\mathbf{s}(x,y)=p$, equation (5) should reduce to the original model probability function $\hat{f}_h(x_n,y_n)$. This sanity check can be easily worked out:

$$\mathbf{P}(y_n|x_n,h,\mathbf{s})=\frac{f_h(x_n,y_n)\mathbf{s}(x_n,y_n)}{\sum_{y\in Y}f_h(x_n,y)\mathbf{s}(x_n,y)}=\frac{f_h(x_n,y_n)p}{\sum_{y\in Y}f_h(x_n,y)p}=\hat{f}_h(x_n,y_n)$$

References

[1] Blais, B. S. (2014). Statistical Inference for Everyone (sie).

[2] Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, B. D. (2013). Bayesian Data Analysis. 3rd edition.

[3] Hoffman, M. D., & Gelman, A. (2014). The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. J. Mach. Learn. Res., 15(1), 1593-1623.

[4] King, G., & Zeng, L. (2001). Logistic regression in rare events data. Political analysis, 9(2), 137-163.

[5] LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., & Huang, F. (2006). A tutorial on energy-based learning. Predicting structured data, 1(0).

[6] Maalouf, M., & Siddiqi, M. (2014). Weighted logistic regression for large-scale imbalanced and rare events data. Knowledge-Based Systems, 59, 142-148.

[7] Maalouf, M., & Trafalis, T. B. (2011). Rare events and imbalanced datasets: an overview. International Journal of Data Mining, Modelling and Management, 3(4), 375-388.

[8] Martin, O. (2016). Bayesian analysis with python. Packt Publishing Ltd.

[4]This makes normalization unneccesary in the final form, which is a very desirable property to have. Specifically, it allows us to bypass cases where normalization is intractable or even consider improper forms.

[9] Salvatier, J., Wiecki, T. V., & Fonnesbeck, C. (2016). Probabilistic programming in Python using PyMC3. PeerJ Computer Science, 2, e55.

[10] Sklar, M. (2014). Fast MLE computation for the Dirichlet multinomial. arXiv preprint arXiv:1405.0099.

[11] Sklar, M., Stewart, R., Li, R., Bakula, A., & Spears, E. (2020). U.S. Patent Application No. 16/405,481. [Section 0037]