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

Max Sklar

November 4, 2021

Abstract

Given a supervised machine learning problem where the training set has been subject to a known sampling bias, we ask if we can still find a model that best fits the original dataset. While information is lost under sampling, we derive an altered formula for the Bayesian posterior distribution to do just that. This is achieved though the Bayesian framework for supervised machine learning, followed by the introduction of a sampling function along with a derivation of an altered posterior distribtion. This generic bias correction formula is then applied to the common case of binary logistic regression. We also discuss scenarios where a dataset might be subject to intentional sample bias, including the case of rare events. Finally, we discuss several different sampling methods, including those that aren't covered by this framework but offer opportunities for generalization.

1 Introduction

A central task in computer science is to develop *algorithms*, or a series of precise instructions, to computing any given *function* $f : X \to Y$. The goal of *supervised machine learning* on the other hand, is to *learn* to compute f through example rather than direct instructions. The examples are given in the form of *instances* of input-output pairs $(x, y) \in X \times Y$ known as the *training set*. The training set that allows the machine to learn the *concept* of f by producing an approximation of it known as a *model*.

Machine learning works best when the training set approximates real world conditions. Engineers must sometimes rely on the fact that models can perform well on data with different compositions from the training set, but that assumption is far from certain to hold.

Occasionally the full dataset of instances is not available either by design or circumstance, and all that remains is a biased sample. **Fortunately, when the sampling function is known, there is a method to learn the concept from the remain evidence available designed for the *original* dataset.**

The method becomes apparent when analyzed through the *Bayesian framework* for supervised machine learning in section 3. Key to this framework is the *posterior probability distribution* over potential models, which allows a search algorithm to discriminate between them by their projected performance. When a bias sample is provided, the posterior distribution formula can be properly corrected as detailed in sections 4 and 5 to counteract this bias.

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 the entire training set. Therefore, sampling or removing data from the training set is a legitimate design decision.

One way to slim down the training set 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 is to train an image recognition algorithm to find images of lions. Furthermore, the training data contains 2000 images of lions, and 400,000 images that are **not** lions. The following 3 sampling schemes 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)

These 3 options each come with tradeoffs.

Option A uses all of available 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 any additional data can only refine our search. In practice this might not be the case because the model search algorithm is imperfect. For example, if a *mini-batch* system is used to find gradients and search for the best model, it might not converge as easily with an imbalanced dataset.

Option B has all the imbalance of A, but with fewer instances. The algoithm will run more quickly, but the results are likely to be worse. To learn lions, and classifier will rely on lion photos, and they have become quite scarce! Because of the imbalance, each lion photo is much more valuable to our success than a non-lion photo. 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 the sampling bias, it will use learn that Lions are more common than they really are in the underlying dataset, and will therefore tend to overpredict lions. Hopefully the underlying image recognition algorithm will derive 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 other cases it is crucial!

An iteration of Foursquare's attribution model[11] provides a clear example. As part of a product that measures the ability of an advertisement to drive consumers to physical locations such as a retail chain, the Foursquare data pipeline first learns the base probability that any given individual would visit that chain on a given day.

Foursquare's data set has many examples of visits, but the amount 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. This was ultimately achieved through the bias correction formula for logistic regression in section 6.

2.3 Similar and Adjacent Work

The literature on sampling and rare events is 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 techniques for it's mitigation, see the work of Maalouf and Trafalis[7]. The readjustment formulas 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 Supervised Machine Learning Framework

The following is a typical setup for 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.

3.1 Given Parameters

Let X be the input space and Y be the output space. The goal is to predict a *label* $y \in Y$ from a corresponding input $x \in X$, or to learn a function $f : X \to Y$.

The training dataset $D \in (X \times Y)^N$ consists of N examples of input-output pairs $(x, y) \in (X \times Y)$. D may be generated by an oracle which produces an arbitrary number of examples, or it might be a small and limited collection. In any case, inference will be based off of just these N examples.

Instances are labelled with a subscript (x_n, y_n), where $n \in \{0, 1, 2, \ldots, N-1\}$, or $n \in N$ in ordinal notation.

3.2 Hypothesis Space

Let H be the *hypothesis space* whose members $h \in H$ each encode a potential solution to the prediction problem. We assume that one of these solutions is correct. In practice we must consider that this space may not contain our solution, but this temporary assumption makes Bayesian inference possible.

Each $h \in H$ is considered a *hypothesis* for a solution to our problem. They are also called *predictors*, *models*, or *solutions*. We will use the term predictor to empasize it's ultimate purpose while using the letter h.

In some setups, the predictors $h \in H$ are *directly predicting* $y \in Y$ and represented by a function from X to Y. Here instead the predictors will return a *probability measure* over Y, which means that $h \in H$ is now *probabilistically predicting* $y \in Y$. While it is possible to generalize to all probability measures over 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 returns 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.

In each of these cases, the model assigns a number to each $y \in Y$. The case of discrete Y it is a finite probability and in the case of continuous Y it is the value of the PDF. If these numbers are normalized, the discrete probabilities will add to 1 and the continous probabilities will integrate to 1.

We do not require these values to be normalized. Instead we say that each predictor h is identified with a *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 $\hat{f}_h : X \times Y \to \mathbb{R}^+$.

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

3.3 Deriving the Posterior Distribution from Bayes Rule

The *prior distribution* over H is a probability measure representing the initial 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 $h \in H$. This may denote a discrete probability or a value in a PDF in the continuous case.

$\mathbf{P}(D|h)$ is the *likelihood function* which denotes the probability of recieving the entire training set D under a given predictor h. This is equal to the product of the probabilities of recieving each label y_n independently.

$$\mathbf{P}(D|h) = \prod_{n \in N} \hat{f}_h(x_n, y_n) \tag{2}$$

$\mathbf{P}(h|D)$ is the *posterior distribution* over H. This probability measure represents the probability of each predictor being correct **after** the data has been taken into account.

Bayes rule for discrete H finds the posterior probability of individual predictor h, and in the continuous case it produces PDF values.

$$\mathbf{P}(h|D) = \frac{\mathbf{P}(D|h)\mathbf{P}(h)}{\sum_{h \in H} \mathbf{P}(D|h)\mathbf{P}(h)} \qquad \mathbf{P}(h|D) = \frac{\mathbf{P}(D|h)\mathbf{P}(h)}{\int_{h \in H} \mathbf{P}(D|h)\mathbf{P}(h)} \tag{3}$$

Because most learning algorithms only require unnormalized values for $\mathbf{P}(h|D)$, we rewrite the equality as a proportionality statement and remote the denominator.

$$\mathbf{P}(h|D) \propto \mathbf{P}(D|h)\mathbf{P}(h) \tag{4}$$

The likelihood of the dataset under predictor h, or $\mathbf{P}(D|h)$, is equal to product of the absolute probability function of each instance. We rewrite the formula for the relative posterior distribution as

$$\mathbf{P}(h|D) \propto \left(\prod_{n \in N} \hat{f}_h(x_n, y_n) \right) \mathbf{P}(h). \tag{5}$$

3.4 Selecting and Sampling Predictors

The final learning task is either to *select* or *sample* a predictor that best explains the data. This process involves a search of H is often identified as the learning algorithm.

In the case of selection, the goal is to identify a single optimal predictor. A good example is the *maximum a posteriori* (MAP) estimate which seeks to find the predictor with the highest posterior probability. Another example is the *maximum likelihood estimate* (MLE) which finds the predictor that assigns the highest likelihood to the dataset.

Under sampling, a predictor is randomly pulled from a distribution approximating the posterior distribution. Unlike selection, this method accounts for the uncertainty inherent in the result. If several models are sampled we can see how much variety there is in the number of possible solutions still consistent with our data.

In general, the hypothesis space H is complex and a variety of methods for selection and sampling have been developed. These techniques include hill climbing and gradient descent for selection, and Markov Chain Monte Carlo for sampling. A good example of the latter is the *No U-Turn Sampler*[3] popular in the PyMC3[9] probabilistic programming package for python. This author has also deployed the Newton Raphson method for gradient descent to calculate the MLE.[10]

For most of these algorithms, it is more convenient to work with a *negative log-likilihood loss* function than the posterior distribution directly.[1] This is because the negative-log likelihood turns products into sums, and produces values that are within a reasonable order of magnitude for computation. We get this by applying $-\ln(\ldots)$ to the right hand side of equation 5.

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

Assume each hypothesis comes with its own negative log-likelihood loss function l_h where $\hat{f}_h(x_n, y_n) \propto e^{-l_h(x_n, y_n)}$ 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

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

4 The Sampling Problem

Consider the case where the training set D was 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 $\mathbf{s} : X \times Y \to [0, 1]$ that consider each datapoint $(x, y) \in D^+$ independently.

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

When the sampling function is known, a posterior distribution can still be computed from D to learn which predictors are more likely to work on D^+.

5 The General Solution

Start with the unnormalized Bayes rule in equation (4), 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

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

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

Note that the expression $\mathbf{P}(y_n|x_n, h, \mathbf{s})$ conditions on x_n, predictor h and sampler $\mathbf{s}$. This is because the predictors tell us only how y_n is produced from x_n and not how x_n is distributed in the first place. The question therefore comes down to the probability of producing y_n. The following *generative description* is a useful tool in understanding how the sampling function factors into the generation of 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 with candidate y comes to

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

If Y is discrete, the probability of selecting candidate $y \in Y$ is $\hat{f}_h(x_n, y)$. Use this to sum over the probabilities of selecting each possible candidate and setup the recursive equation

$$\mathbf{P}(y_n|x_n, h, \mathbf{s}) = \sum_{y \in Y} \hat{f}_h(x_n, y)\big(\mathbf{s}(x_n, y)\,[y_n = y] + (1 - \mathbf{s}(x_n, y))P(y_n|x_n, h, \mathbf{s})\big). \tag{6}$$

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 formulas for both discrete and continuous[2] Y.

$$\mathbf{P}(y_n|x_n, h, \mathbf{s}) = \frac{f_h(x_n, y_n)\mathbf{s}(s_n, y_n)}{\sum_{y \in Y} f_h(x_n, y)\mathbf{s}(x_n, y)} \qquad \mathbf{P}(y_n|x_n, h, \mathbf{s}) = \frac{f_h(x_n, y_n)\mathbf{s}(s_n, y_n)}{\int_{y \in Y} f_h(x_n, y)\mathbf{s}(x_n, y)} \tag{7}$$

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].

[2] The continuous version of this argument requires more mathematical background but is completely analogous. It requires integrating overover all candidates instead of summing, and using a dirac measure instead of the indicator function.

[3] small enough to enumerated by the machine that is doing the computation

5.1 Formula for Negative Log-Likelihood

Equation (7) provides all the tools needed to assign relative posterior probabilities to predictors. We derive a negative log likelihood loss function for selection or sampling 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 (7) to get this in terms of f_h:

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

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

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

6 Solution for Binary Logistic Regression

Binary logistic regression is a special 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.

Each hypothesis $(c,\mathbf{w}) \in H$ 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 (7) we get

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

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 biased sampling, and would correspond to the lion image in section 2.1 and the visit in section 2.2. We can solve for it 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 correct for sampling bias in all binary classification. Note that for $r_n = 1$, we are left with the original 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 $\mathbf{w}_f$ or intercept c can be computed into the following simple form in order to perform gradient descent.

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

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

7 Future Work

7.1 Simple Random Sampling

Simple Random Sampling (SRS) is where an exact number of datapoints is retained in the dataset. The point-wise sampling function can never guarantee a specified number of instances will be selected. Simple Random Sampling can furthermore be stratefied by output to account for rare events. Because the sampling process is no longer independant by instance, deriving the formula is far more complex.

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 this is often the *poisson distribution*. 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 The Value of an Instance

Finally, there is much work to be done in quantifying the expected value that an instance will bring if included in D. The complication here is that we don't know how much a datapoint will change the posterior distribution 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 estimate the value that an instance provides versus the cost of including it would eliminate unneccary work.

Additionally, several 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, and each part runs quickly with a small sample size, this provides real engineering benefits.

Appendices

A Solving for the General Formula

Here we start with equation (6) and go through the series of steps neccesary to derive its final form in equation (7). Equation (6) begins 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

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

In the first sum, the only non-zero addend is $y = y_n$. Therefore, we can replace y with y_n and remove the summation and indicator function. 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 distribute $\hat{f}_h(x_n, y)$ in 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. Thus with one extra step of division, the final form can be given as (7).

$$\mathbf{P}(y_n|x_n, h, \mathbf{s}) = \frac{f_h(x_n, y_n)\mathbf{s}(s_n, y_n)}{\sum_{y \in Y} f_h(x_n, y)\mathbf{s}(x_n, y)} \tag{7}$$

If we assume that there was no sampling, or the sampling was uniform where $\mathbf{s}(x, y) = p$, equation (7) should reduce to the original predictor probability function $\hat{f}_h(x_n, y_n)$ as a sanity check.

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

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.

[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]