Sampling Bias Correction for Supervised Machine Learning

A Bayesian Inference Approach with Practical Applications

Max Sklar

February 9, 2022

Abstract

Given a supervised machine learning problem where the training set has been subject to a known sampling bias, we still want to train a model to fit the original dataset. This can be achieved by introducing a sampling function to the Bayesian inference formula, and deriving an altered posterior distribution for the task. We then apply this to the common case of binary logistic regression, and discuss scenarios where a dataset might be subject to intentional sample bias such as label imbalance. This technique is widely applicable to statistical inference on big data, from the medical sciences to image recognition to marketing. Familiarity with it will give the reader tools to improve their inference pipeline from data collection to model selection.

1 Introduction

Developing *algorithms* is a central task in computer science. Each algorithm is a series of precise instructions to compute a given mathematical *function* $f : X \to Y$.

This works great when the programmer has these precise instructions available, but not when a function is only partially known, or if it is too complex for even a large team. *Machine Learning* was developed in part as a response to this problem, to extend the ultimate reach of software.

In *supervised machine learning*, an algorithm *learns* to compute a specific function f through example rather than through direct instructions. 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 allows the machine to learn the *concept* of f by producing an approximation of it known as a *model*. Models are often evaluated on a *test set* of instances and finally deployed on real world data to make predictions.

Machine learning is most reliable when the training set approximates these real world conditions. This isn't always possible. Models might perform well anyway, but that assumption is far from certain to hold.

Occasionally an accurate training set is unavailable either by design or circumstance, and all that remains is a biased sample. This could happen in many ways. Databases that contain vast market knoweldge could have had records deleted in a non-random way. Usually datasets and experiments involving humans have sensitive information that will need to be redacted and filtered. Finally - and perhaps most instructively - vast amounts of data might be deemed unhelpful to solving the problem in question, and it could be temporarily discarded in order to save costs.

Fortunately, there is a significant subset of these cases for which this bias is not a problem: **When the sampling function is known, there is a method that can learn the concept from the remaining evidence available, and produce a model to fit the *original* dataset.**

In this paper, we will fully document this method. We analyze the problem through the *Bayesian framework* for supervised machine learning reviewed 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.

Unlike ad-hoc methods for correcting bias after a model has been created, this method bakes the bias-counteracting term into the learning process itself and is shown here to be consistent with first principles. It also has been successfully deployed in major commerical product launches, as documented in section 2.2.

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. Part of this cost is related to the size of the dataset being used to learn models.

This raises a question for the machine learning practitioner: should we use all the data available to us in the training of a model? Should we even take steps to procure additional data that we currently do not have? Sometimes the answer is yes. When data is scarce, each additional piece of information help us learn a better model. But as the amount of data starts increasing by orders of magnitude, this improvement in model fit starts to diminish, and the cost of training it increases. At some point the cost of obtaining and training on more data exceeds the value of the improvements to the model.

There is even a point where additional data adds zero value as model impovements start amounting to a rounding error. This could happen very quickly on simple models (for example, a linear regression in few variables) or it could never happen (perhaps in a deep learning setup). The point is that depending on the situation, the practitioner may choose to remove data from consideration, and this is a legitamate 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. Therefore, we may choose to remove data in a non-random fashion as the following examples illustrate.

2.1 Example: Theoretical Image Recognition

Suppose that the goal is to train an image recognition algorithm to detect lions. The training data contains 2000 images of lions, and 400,000 images that are **not** lions. We choose an approach to modelling these images (perhaps a convolutional neural net) and decide that we may want to sample out some of the data. Consider the following 3 sampling schemes.

(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)

Each option comes with tradeoffs.

2.1 uses all of the available data and takes the most time and resources to train. In theory, this will produce a model that is at least as good as any of the others.

(B) has fewer instances than but with the same label imbalance. The learning algoithm will run faster, but will likely product worse results. Any classifier must rely on lion photos to learn lions, and those have become quite scarce! A reduction in lion photos from 2000 to 500 will make a significant negative impact. Because of this imbalance, each lion photo becomes much more valuable than a non-lion photo.

(C), like (B), demands fewer resources. The difference now is that all 2000 lion photos remain. The likely outcome is that the model from (C) performs far better than (B). It might even perform as well as 2.1! It seems to be the best choice.

Because (C) only removed non-lion images, it will learn that lions are more common than they really are in the underlying dataset. Our sampling method and therefore our derived dataset now have a *bias* towards lions over non-lions. This causes it to overpredict the lions, particularly in those marginal cases where it is uncertain.

Hopefully the underlying image recognition algorithm will derive the core visual features of a lion and will not have that issue even without need for correction, but such a correction is desirable.

2.2 Example: Probabilistic Event Detection

In the image recognition example, bias correction is desirable. In other cases, it is crucial!

An iteration of Foursquare's attribution model[13] provides a clear example. The company's attribution product measures the ability of an advertisement to drive consumers to physical locations such as a retail chain. In order to estimate causality, the Foursquare data pipeline must first learn the base probability that any given individual will visit that chain on a given day.

Foursquare's data set has many examples of visits, but examples of people who "did not visit" on any given day outnumbers visit by several orders of magnitude. Therefore, it makes sense to downsample those non-visits. Because the value of an ad measurement product is in its accuracy, the bias sampling must be accounted for precisely. 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. Rather than providing a comprehensive review, we refer to some selected works which were either helpful or inspirational in the course of this research.

For general cataloging of situations with imbalanced classification and techniques for it's mitigation, see the work of Maalouf and Trafalis[9]. The readjustment formulas in the case of logistic regression can be found

in Maalouf and Siddiqi[8]. 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[6].

3 Supervised Machine Learning

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[10][4][2]. 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 the 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 The 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. This temporary assumption makes Bayesian inference possible.

Each $h \in H$ is considered a *hypothesis* for the correct function f. They are also called *predictors*, *models*, and *solutions*. We will use the term predictor to emphasize its ultimate purpose.

In some setups, the predictors $h \in H$ are *directly predicting* $y \in Y$ and are represented by a function from X to Y. Here instead the predictors will return a *probability distribution* 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 *classification* problem. 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. This is a *regression* problem. 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 $y \in Y$. With discrete Y it is a finite probability,

and with continuous Y, it is the value of the PDF. If these numbers are normalized, the discrete probabilities will add to 1 and the continous PDF will integrate to 1.

Because these values may not be normalized at first, we identify each predictor h with a *relative probability function* $f_h : X \times Y \to \mathbb{R}_{\geq 0}$ which assignes a non-negative real number to each input-output pair. Define the normalized probability function $\hat{f}_h : X \times Y \to \mathbb{R}_{\geq 0}$ as follows for both discrete and continuous Y.

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

3.3 Deriving the Prior, Likelihood, and Posterior

The *prior distribution* over H is a probability distribution representing the initial belief over which predictor is correct. Typically this will be an *uninformative prior* which encodes our 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 is 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.

$\mathbf{P}(h)$ does not need to be normalized. It is enough that it represents a relative probability function on h which preserves the ratio of the probabilities between two predictors. This includes the possibility of an *improper probability distribution function*. For example, a PDF that assigns 1 to each real number if $H = \mathbb{R}$ cannot be normalized, but is not excluded. More importantly, allowing the prior to be unnormalized means that we can use distributions whose integral or sum is difficult to compute.

$\mathbf{P}(D|h)$ is the *likelihood function*. It denotes the probability of recieving the entire training set D under a given predictor h. We assume that the examples in the training set are *independent and identically distributed* (IID). This means that the likelihood 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. The posterior 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 each $h \in H$, and in the continuous case it produces PDF values.

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

Most learning algorithms only require unnormalized values for $\mathbf{P}(h|D)$. We rewrite the equality as a proportionality statement and remove the denominator. This form also allows the prior $\mathbf{P}(h)$ to be unnormalized as well, and generalizes both the discrete and continuous case.

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

Going forward, any terms on the right hand side of this proportionality that are constant with respect to h can be removed. Using equation (2), 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 to either *select* or *sample* predictors that best explain the data. This process involves a search of H and is identified as the search algorithm or learning algorithm.

The goal of selection is to identify a single optimal predictor. The most obvious variable to optimize is the posterior probability (or PDF value) and this is called the *maximum a posteriori* (MAP) estimate. The *maximum likelihood estimate* (MLE) finds the predictor that assigns the highest likelihood to the dataset, ignoring priors altogether. Techniques to find this optimal predictor include hill climbing and gradient descent. The newton-raphson method converges faster than gradient descent and this author has used it to calculate the MLE for the Dirichlet-multinomial problem.[12]

Under sampling, a predictor is randomly pulled from the posterior distribution or something approximating it. If several models are sampled we can obtain a wide variety of possible solutions that are still consistent with the data. Markov Chain Monte Carlo methods are used for sampling. A good example of this is the *No U-Turn Sampler*[5] popular in the PyMC3[11] probabilistic programming package for python.

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

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

Let 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

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

[1] For a great treatment on loss functions and their various tradeoffs, see A Tutorial on Energy Based Learning by Lecun[7]. The work also discusses strategies for dealing with predictors where normalization over Y is not practical thus avoiding $\hat{f}_h$.

4 The Sampling Problem

What if 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, 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 fit 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 $\mathbf{s}$.

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

Let $\mathbf{P}(x_n \in D^+)$ represent the probability that any given input x_n will appear in the unbiased dataset D^+. The allows us to break down the likelihood as follows.

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

The term $\mathbf{P}(x_n \in D^+)$ is constant with respect to h and can therefore be dropped in the proportionality statement. We are left with calculating the expression $\mathbf{P}(y_n|x_n, h, \mathbf{s})$. The following *generative description* is a useful tool in understanding how y_n is produced when there is a sampling function.

1. Consider as given an input x_n, a predictor h, and a sampling function $\mathbf{s}$. The predictor h encodes a probability distribution over Y through the 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.

We now use the generative description to produce a recursive equation for $\mathbf{P}(y_n|x_n, h, \mathbf{s})$. Let $y \in Y$ be the first candidate for y_n. 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 y is not accepted, then the probability reverts to the original value of $\mathbf{P}(y_n|x_n, h, \mathbf{s})$. Putting it together, the probabilty of ultimately accepting y_n 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 y as a candidate in the first place is $\hat{f}_h(x_n, y)$. Use this to sum over the probabilities of selecting each possible candidate and setup a 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 it is at least easy when Y is finite and small enough to be enumerated by a machine.

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 starting with the relative Bayes formula.

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

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:

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

[2]The continuous version of this argument requires more mathematical background but is completely analogous. It requires integrating over all candidates instead of taking the sum. Some care must be taken with the indicator function term for a rigorous argument, but ultimately it can be reduced in the same way.

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

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

Use equation (7) to 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:

$$\mathbf{P}(y_n = 1 | x_n, h, \mathbf{s}) = \frac{e^{c + \mathbf{w} \cdot x_n}\mathbf{s}(s_n, 1)}{\mathbf{s}(x_n, 0) + e^{c + \mathbf{w} \cdot x_n}\mathbf{s}(x_n, 1)} = \frac{e^{c + \mathbf{w} \cdot x_n}}{\mathbf{s}_r(x_n) + e^{c + \mathbf{w} \cdot x_n}} \qquad \text{where} \quad \mathbf{s}_r(x_n) = \frac{\mathbf{s}(x_n, 0)}{\mathbf{s}(x_n, 1)} \tag{8}$$

Here, $\mathbf{s}_r : X \to [0, \infty]$ is the true-to-false *sample ratio* for each instance n. This ratio is all that is needed to correct for sampling bias in any binary classification. Note that for $\mathbf{s}_r(x_n) = 1$, 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.

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

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.

$$\frac{\partial}{\partial \mathbf{w}_f} L(h) = \sum_{n \in N} x_{n,f} \left(\frac{e^{c+\mathbf{w}\cdot x_n}}{\mathbf{s}_r(x_n) + e^{c+\mathbf{w}\cdot x_n}} - y_n \right) + \lambda \cdot w_f \qquad \frac{\partial}{\partial c} L(h) = \sum_{n \in N} \left(\frac{e^{c+\mathbf{w}\cdot x_n}}{\mathbf{s}_r(x_n) + e^{c+\mathbf{w}\cdot x_n}} - y_n \right) \quad (9)$$

7 Future Work

7.1 Simple Random Sampling

Our sampling function $\mathbf{s}$ decides on the inclusion of each training instance independently. Such a sampling function can never guarantee a exactly specified number of instances will be selected. When an exact number of datapoints are retained, this is called *Simple Random Sampling* (SRS). Simple Random Sampling can be stratefied by label to correct for the imbalance of rare events. Because the sampling process is no longer independent by instance, deriving the posterior formula is now far more complex task.

7.2 Oversampling

The sampling function 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. *Bootstrap sampling* methods rely on oversampling.

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 it is often the *poisson distribution*. The poisson distribution with parameter λ places probability of multiplicity k at $\frac{\lambda^k e^{-\lambda}}{k!}$.

7.3 The Value of an Instance

The ability to quantify and quickly estimate the expected value that an instance will provide if included in D would allow engineers to deploy samplers that eliminate unneccesary work.

Perhaps insights on sampling can be borrowed from the field of *active learning*, where learners can directly choose inputs to query. Angluin[1] provides foundational analysis on the efficiencies that can be achived with active strategies. Many such strategies have been deployed for statistical models, and for example Cohn et al.[3] propose selecting input data to minimize learner variance.

Ultimately, we do not know in general how much a datapoint will change the posterior distribution before it is included, but estimation techniques could be developed. Engineers must also consider the specific goals and tradeoffs of a project to inform their choice of sampling strategy.

7.4 Stochastic Gradient Descent and Mini-Batch Training

Many machine learning algorithms choose to look at training instances one at a or in small batches (mini-batch) instead of all at once. These algorithms can also incorporate bias data selection and bias correction.

7.5 Ensemble Models

If models produced from several sampling functions can be combined into 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.

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

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

Fom equation (1) we know that for all possible inputs x, $\sum_{y\in Y} \hat{f}_h(x,y) = 1$. We can simplify as follows:

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

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

Also from equation (1), the $\hat{f}_h$ terms are simply a constant factor of same terms with the unnormalized version f_h. Because this 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 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] Angluin, D. (1988). Queries and concept learning. Machine learning, 2(4), 319-342.

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

[3] Cohn, D. A., Ghahramani, Z., & Jordan, M. I. (1996). Active learning with statistical models. Journal of artificial intelligence research, 4, 129-145.

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

[5] 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.

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

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

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

[9] 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.

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

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

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

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