Sampling Correction for Supervised Machine Learning
A Probabilistic, Bayesian Approach

Max Sklar

August 7, 2021

Abstract

ABSTRACT - THIS IS AN UNFINISHED DRAFT - DO NOT PUBLISH

1 Introduction

The goal of *supervised machine learning* is to develop, execute, and deploy algorithms which learn to compute functions through example rather than being given instructions to compute these functions directly. This type of machine learning algorithm learns to approximate this 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 "looks" like the training set, as if these *instances* were all pulled from the same source producing the same distribution. Sometimes models can perform well on datasets with somewhat different compositions than their training set, and in practice we do see this relied upon from time to time. However, it's an assumption that is far from certain to hold.

There are many practical examples of cases where a full underlying dataset, whether by design or circumstance, is not available, and only an unevenly sampled subset is. **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 design 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, and each model gives not a direct output to any input $x \in X$, but a probability distribution over possible outputs $y \in Y$. We are then able to use the dataset to compute a *posterior distribution* over the potential models, allowing us to discriminate models 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. The formulas may be more complicated by the inclusion of a sampling function but 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 useful 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 models 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 being in the sampled set. This process ignores the fact that certain training examples are worth more than others, particularly when there is a classification imbalance.

2.1 Example: Image Recognition

Suppose that the goal was to train an image recognition algorithm to recognize the image of a lion. Suppose that the training data contains 2000 images of lions, and about 400,000 images that are **not** lions. The following 3 schemes for sampling the data 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 therefore will take 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 find 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 has less data to consider. The model space can be searched faster under 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 big 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 it'll be just as easy to run as 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. In this case, 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

The need to account for uneven sampling becomes far more critical in 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 has a pending patent[6]... As part of a product that measures the ability of an online advertisement to drive consumers to the stores of a given retail chain, the Foursquare data pipeline first had to build a model that calculates the probability that any given individual would visit that chain on any given day.

Foursquare has access to a visit database with 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 Previous Work

Final Example: Maalouf paper previous work
mention most bibliography items here
Yan Lecun' Energy Paper

3 Bayesian Supervised Machine Learning Framework

The following is a typical setup for probabilistic supervised meachine learning using a Bayesian framework. The terminology and variable names will be reused in subsequent sections of this paper.

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 $h \in H$ which each encode a 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

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

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 in this paper 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.

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 probability distribution function.

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

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

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

Once H has been defined, the task is either to *select* a model or *sample* a predictor that best explains the data. In the case of selection, the analysis provides a a single predictor whereas in sampling, a series of predictors are randomly generated according to their relative strength in explaining the data.

We again rely on the concept of *relative probability*, this time between two potential models $h_1, h_2 \in H$ and their predictive power. In general, the hypothesis space H is very complex and a variety of methods for selection and sampling have been developed. [SOURCE] Source: MLE Paper from Max, NUTS MCMC Paper.

The final component to a Bayesian analysis of the problem is the *prior distribution* over H which is a probability measure representing the current belief over which predictor $h \in H$ 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 here use $\mathbf{P}(h)$ as the *prior probability* of predictor $h \in H$.

Bayes rule for discrete H finds the probability of individual predictor h_i. $\mathbf{P}(D|h)$ is the probability of seeing the outputs in dataset D under the model h

$$\mathbf{P}(h_i|D) = \frac{\mathbf{P}(D|h_1)\mathbf{P}(h_1)}{\sum_{h\in H}\mathbf{P}(D|h)}$$

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

$$\mathbf{P}(h|D) \propto \mathbf{P}(D|h)\mathbf{P}(h) \tag{2}$$

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

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

It is often more convenient to work with a *negative log-likilihood loss* function 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 more convenient for model selection or sampling through techniques such as hill climbing, gradient descent, or Markov Chain Monte Carlo. [Include some citations] [Should I introduce a catch-all term for "selection or sampling"?]

4 The Sampling Problem

In the previous setup, D is a raw, unsampled dataset. Now consider cases where D has been derived by downsampling a larger dataset D^+. Formally, we say that D was generated from D^+ with a sampling function $\mathbf{s} : X \times Y \to [0, 1]$. For each element $(x, y) \in D^+$, the sampling function $\mathbf{s}(x, y)$ returns the probability that this example will be in D.

Given access to a way to computer $\mathbf{s}(x, y)$ and D^+, one could sample and generate a smaller dataset. This is how we assume D was created. Now from just the dataset D and knowledge of $\mathbf{s}$ we calculate the posterior probability and loss function in order to correctly apply Bayesian inference.

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 the remaining data. The sampling affects what that final loss function knows about the data, but the results will represent the best-guess from the data available.

5 The General Solution

Start with the unnormalized Bayes rule in equation (2), 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 to

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

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 mental model is a useful tool in understanding how the sampling function s factors into the generation of these labels y_n under any predictor $h \in H$.

1. Consider as given 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 $f_h(x_n, y_n)$ or loss function $l_h(x_n, y_n)$.
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 or not 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 equal to 0 for every potential candidate of any given x_n. This is a fair assumption because if it were false, x_n would never appear in the dataset D.

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

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 landing on y_n comes to

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

Assume that Y is discrete, we can use the normalized probability function of model h given in equation (1) to sum over all possible candidates y and setup the recursive equation

$$\mathbf{P}(y_n|x_n,h,\mathbf{s}) = \sum_{y\in Y} \hat{f}_h(x_n,y)\big(\mathbf{s}(x_n,y)\,[y_n = y] + (1-\mathbf{s}(x_n,y))P(y_n|x_n,h,\mathbf{s})\big). \tag{3}$$

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

$$\mathbf{P}(y_n|x_n,h,\mathbf{s}) = \frac{f_h(x_n,y_n)\mathbf{s}(s_n,y_n)}{\sum_{y\in Y} f_h(x_n,y)\mathbf{s}(x_n,y)}. \tag{4}$$

For a continuous output space Y, the following analogous equation can be derived using integrals rather than a sums.

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

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 enough to enumerated by a machine. The formula generalizes to more exotic probability measures on Y, though the rigorous treatment of this is outside our scope.

5.1 Formula for Negative Log-Likelihood

Equation (4) 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 relative Bayes.

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

Then, use equation (4) 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 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* problem means that the learning problem in section **??** 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, $X = \Re^{|F|}$.
3. Each predictor $h \in H$ is parameterized by $h = (c, \mathbf{w})$ where $c \in \Re$ is the intercept and $\mathbf{w} \in \Re^{|F|}$ is an $|F|$-dimentional vector of weights corresponding to each input feature.

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 (4) and simplifying we see

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

Occasionally these problems are framed to focus on the probability that the *target condition* is met, or $y_n = 1$. Often, this target condition is the rare event that incentivized the use of adaptive sampling to begin with! 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, $r_n = \mathbf{s}(s_n, 0)(\mathbf{s}(s_n, 1))^{-1}$ 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 usual 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 to the following simple form, which is required for gradient descent and some versions of Markov Chain Monte Carlo.

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

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

7 Future Work

Concept of the marginal Value of a Data Point

Appendices

A Solving for the General Formula

Here we start with equation (3) and go through the series of steps neccesary to derive its final form in equation (4). Equation (3) beings as follows:

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

Break out the summation as follows:

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

Note that in the first summation, the only non-zero addend will occur when $y_n = y$ and the corresponding indicator function is 1. Therefore, we can replace y with y_n and remove the summation and the indication function.

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

Now 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. Thus with one extra step of division, the final form can be given as (4).

$$\mathbf{P}(y_n|x_n,h,\mathbf{s})=\frac{f_h(x_n,y_n)\mathbf{s}(s_n,y_n)}{\sum_{y\in Y}f_h(x_n,y)\mathbf{s}(x_n,y)} \tag{4}$$

If we assume that there was no sampling, or the sampling was uniform where $\mathbf{s}(x,y)=p$, equation (4) 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] Robert, C. (2007). The Bayesian choice: from decision-theoretic foundations to computational implementation. Springer.

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

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

[4] 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.

[5] Wallach, H. M. (2008). Structured topic models for language. Unpublished doctoral dissertation, Univ. of Cambridge.

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