Fast Confidence Estimation for Classification and Regression models

Andrei Stroganov, Andrei Visochan

Samsung Research

1. The prediction confidence

Why do we need it? How to define it?

The problem

Prediction quality estimation plays key role in model tuning and there are many metrics (F1, ROC/PR AUC, etc) to do such estimation. During tuning we usually have testing dataset, so the true values are known and we can compare the predicted values with them.

There are scenarios when ML model is applied on a group basis, for instance: detect the infected persons within a group. In this case we might be interested not only how well the model performs on individual samples, but also — on the entire groups. But what if we don't have the true values for validating the predictions? Can we tell if group predicted "probably ok" or predictions seems improbable?

As a motivating example for this problem we'll consider ML method for Profile Guided Optimization (PGO) that we proposed in [1].

[1] A. Visochan, A. Stroganov, I. Titarenko, S. Lonchakov, S. Mologin, S. Pavlova, A. Lyupa, A. Kozlova, "Method for Profile-Guided Optimization of Android Applications Using Random Forest," in IEEE Access, vol. 10, pp. 109652-109662, 2022, doi: 10.1109/ACCESS.2022.3214971.

A motivating example

The Android applications consist of Java bytecode of classes and methods, which can be compiled into native representation usually resulting in faster application startup and smoother performance. In PGO not all classes and methods are compiled, but only those which are frequently used (hot), their indexes are listed in a special file — a profile.

In ML approach [1], a model predicts hot items and creates an optimization profile on a per-application basis. There are cases when testing profile is unavailable, so direct prediction validation is impossible. In this talk we'll discuss a p-value based method to indirectly estimate the prediction confidence in similar cases.

To which predictions do we trust?

How many hot methods in given application?

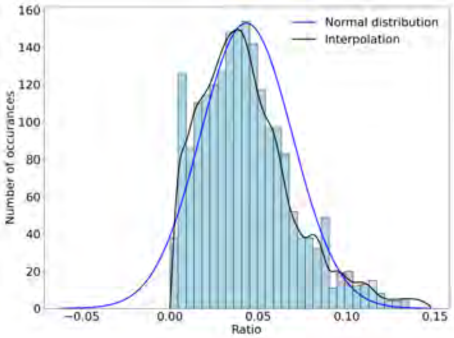
Application	**Number of methods**	**Predicted hot**
Solitaire	295019	7905
Downlood	76791	8433
Sofascore	166963	16151
Connect	383885	15813
Fontboard	68180	2083

Our goals:

- Provide a metric which evaluates a confidence level as real number between 0 and 1, where 1 is most probable result, 0 is most improbable;
- Metric evaluation should be time-efficient;
- Implementation should be easily verified (for stability and undefined behavior safety).

Examine the real data distribution

	Total	Hot	Ratio Hot/Total
1	277512	11604	0.042
2	136652	3886	0.028
3	161363	6255	0.039
4	375659	16023	0.043
5	300533	7316	0.021
6	330094	12063	0.037
7	202537	17172	0.085
8	521079	25066	0.048
9	104607	9613	0.092
10	94947	2890	0.030
...	...	...	...



2. Defining a metric

Recall: a probability distribution function

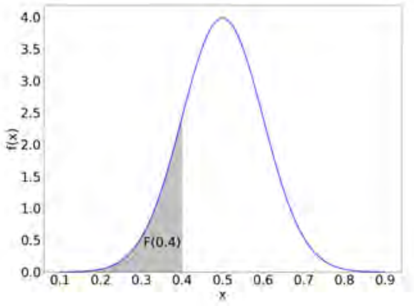
Probability Distribution Function (PDF) $f(x)$ describes how probabilities are assigned to the possible outcomes of a random variable.

Cumulative Distribution Function (CDF) $F(x)$ gives the probability that the random variable is less than or equal to a certain value:

$$F(x) = P(X \leq x)$$

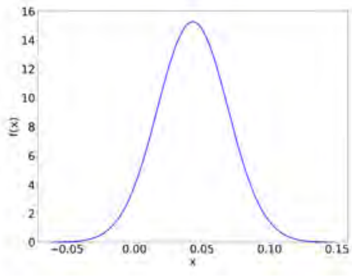
- $F(x)$ is non-decreasing: if $a < b$ then $F(a) \leq F(b)$.
- $F(x)$ is bounded: $0 \leq F(x) \leq 1$.
- $F(x)$ is defined as integral of $f(x)$:

$$F(x) = \int_{-\infty}^{x} f(t)dt$$



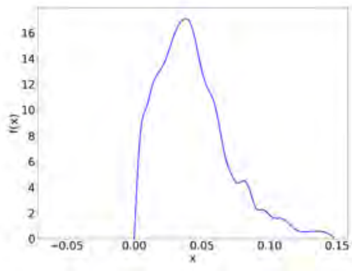
PDF approximation

From the data distribution we get mean and standard deviation values: $\mu \approx 0.043$, $\sigma \approx 0.026$.



$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

A spline approximation of experimental data



A confidence measure

We will use a p-value based approach [2].

Let t be a value for which we measure confidence, $f(x)$ is a PDF and $F(x)$ is a CDF.

- $F(t) = P(x \leq t)$ is a probability that a randomly picked f-distributed value is less or equal than t.
- $1 - F(t) = P(x > t)$ is a probability that a randomly picked f-distributed value is greater than t.

If $F(t)$ and $1 - F(t)$ differs significantly, then t is shifted from the most of the values, such value is unlikely to occur. Thus we define confidence metric $\varphi(t)$ as:

$$\varphi(t) = 2 \cdot \min\left(F\left(t\right), 1 - F\left(t\right)\right)$$

Note that $\varphi(t) = 1$ when there are equal chances of randomly picked value to be greater than t or less than t.

[2] 2004, Lavine, M., Introduction to Statistical Thought, free internet publication

So, how confident we are about predictions?

App	Number of methods	Prediction	Confidence
Solitaire	295019	7905	0.53
Download	76791	8433	0.01
Sofascore	166963	16151	0.04
Connect	383885	15813	0.94
Fontboard	68180	2083	0.63

3. Approaching the computation of $\varphi(x)$

Computing of $\varphi(x)$: a plan

Recall the goals

- Create a metric which evaluates a confidence level as real number between 0 and 1, where 1 is most probable result, 0 is most improbable;
  Done:
  $$\varphi(t) = 2 \cdot \min\left(F\left(t\right), 1 - F\left(t\right)\right)$$
  Note: Usually this requires integration.
- Metric evaluation should be time-efficient;
- Implementation should be easily verified (for stability and undefined behavior safety).

Computing of $\varphi(x)$: time-efficient

To compute $F(x)$ we will use a numerical integration on evenly distributed grid with step size Δ. The integration range is defined by PDF, let it be $[a, b]$, thus we consider $f(x)$ to be negligibly small for $x < a$ or $x > b$.

Prepare a lookup table. Let's store the values of the integral

$$\int_a^x f(t)dt$$

at points $a, a + \Delta, a + 2\Delta, \dots, b$ in table `cdf`:

$$cdf[i] = \int_a^{a+i\cdot\Delta} f(t)dt$$

for $i = 0, \dots, n$, where $a + \Delta \cdot n = b$. Note that $cdf[0] = 0$ and $cdf[n] \approx 1$.

Query in runtime in O(1).

```
float phi(float x) {
    if x <= a or x >= b: return 0

    i = int((x - a) / Delta)
    return 2 * min(cdf[i], 1 - cdf[i])
}
```

Here we provide a simplified pseudo-code. Our C++ implementation is more general, it is available at: `https://github.com/savthe/prediction-confidence`.

Computing of $\varphi(x)$: Implementation should be easily verified

There are several approaches to generating a lookup table with its pros and cons:

- Generate a table when app starts;
  Pros: fairly simple.
  Cons: increases start time, additional runtime code and unit-testing.
- Generate table before app starts and initialize it with every single value;
  Pros: fairly simple, lesser unit-testing.
  Cons: may be difficult for support if we decide to change μ and σ.
- Generate table before app starts using C++ feature **Generic Programming**.
  Pros: table is generated at compile time. The runtime code is very simple, μ and σ can be changed without modifying runtime code. We can test values at compile time, so lesser unit-testing required
  Cons: code is more complex than runtime solution.

We will implement table generation with generic programming.

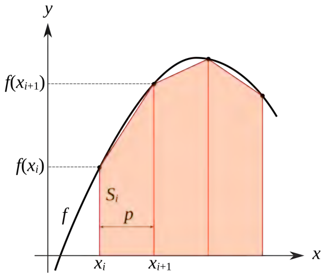
4. The implementation

Computing the integral

We will use the trapezoid rule[3] for approximating the integral

$$\int_a^b f(x)dx$$

we define the grid of size Δ with points $x_1 = a, a+\Delta, \dots, b = x_n$. The area between each consequent points x_i, $x_{i+1} = x_i + \Delta$ is approximated by trapezoid.



$$s_i = (f(x_i) + f(x_{i+1})) \cdot \frac{\Delta}{2}$$

$$\int_a^b f(x)dx \approx \Delta \cdot \left(\frac{f(x_1) + f(x_n)}{2} + \sum_{i=2}^{n-1} f(x_i) \right) \qquad (1)$$

[3] https://en.wikipedia.org/wiki/Trapezoidal_rule

Computing the integral: a lookup table

Denote $x_i = a + \Delta \cdot i$, and let

$$cdf[i] \approx \int_a^{x_i} f(t)dt \approx \Delta \cdot \left(\frac{f(a) + f(x_i)}{2} + \sum_{k=2}^{i-1} f(x_k) \right)$$

The table is filled with each $f(x_k)$ computed only once:

```
cdf[0] = 0;
sum = f(a) / 2
for i in [1, n]:
    f_i = f(a + i * Delta)
    cdf[i] = Delta * (sum + f_i / 2)
    sum = sum + f_i
```

Computing the PDF for Normal distribution

The probability distribution function $f(x)$ is given by:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \qquad (2)$$

The constant part is easy to compute $\left(\sqrt{2\pi} \approx 2.50662827463\right)$, but standard $exp(x)$ function is not available at compile time (it is not a constexpr function in C++17). So, we make one ourselves! Recall the Taylor expansion of e^x:

$$e^x = 1 + x + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} + \ldots \qquad (3)$$

The straightforward implementation would be:

```
accuracy = 0.000001
e = 0
term = 1
i = 1
while term > accuracy:
    e = e + term
    term = term * x / i
    i = i + 1
return e
```

But there is a problem: term $x^n/n!$ can become quite large before factorial scales it down to zero, this leading to overflow. We'll use this code for $|x| < 1$.

Computing e^x for $|x| \geq 1$

Let $x = u + v$, where u is an integer part of x, and v is a fractional part of x, and

$$e^x = e^{u+v} = e^u \cdot e^v. \tag{4}$$

We already have implementation for e^v. For e^u we will use binpow[4] algorithm. Suppose $u \geq 0$, if not, we will apply algorithm to $1/e \approx 0.36787944117$.

```
x = 2.718281828459045 // This is exp(1)
p = 1
while u > 0:
    // if u is odd
    if u % 2 == 1:
        p = p * x

    x = x * x
    u = u / 2
return p
```

Combination of two provided methods allows to efficiently compute $exp(x)$ in compile time.

[4] https://cp-algorithms.com/algebra/binary-exp.html

Conclusion

- We discussed a confidence estimation approach for classification and regression models based on p-value score.
- Provided an implementation of time complexity $O(1)$ (`https://github.com/savthe/prediction-confidence`).
- The approach is general and can be applied to various data distributions, including the approximations of experimental data.

Thank you for your attention