Fast – Asymptotically Optimal – Methods for Determining the Optimal Number of Features

Saeid Tizpaz-Niari[1], Luc Longpré[1], Olga Kosheleva[2], and Vladik Kreinovich[1]

[1,2]Departments of [1]Computer Science and [3]Teacher Education
University of Texas at El Paso, El Paso, Texas 79968, USA
saied@utep.edu, longpre@utep.edu, vladik@utep.edu, olgak@utep.edu

1. Selecting optimal number of features: an important problem

- In machine learning – and in data processing in general – an important problem is selecting the number of features.
- When we only use very few of the available features:
  - the results are not very good,
  - since we do not use a significant portion of the available information.
- As we increase the number of features, the results get better and better.
- However, at some point, we exhaust useful features.
- So, we start adding features that practically do not contribute to the desired decision.
- In such situations, new features mostly adds noise, so the performance deteriorates again.

2. Selecting optimal number of features (cont-d)

- In other words, the effectiveness E depends on the number of features n as follows:
  - the value $E(n)$ first increases with n,
  - but at some point, it starts decreasing with n.
- We need to find the value n_0 at which the effectiveness $E(n)$ is the largest.

3. How this problem is solved now

- Of course, we can always do it if:
  - first, we order the features in terms of their prospective usefulness,
  - e.g., by the absolute value of the correlation between this feature and the value that we are trying to predict, and then
  - we add features one by one in this order, and select the number of features that leads to the best prediction.
- This is usually how people solve this problem now.

4. A better method is needed

- In many cases – e.g., for machine learning – the adding-features-one-by-one algorithm is very time-consuming.
- Indeed, for each number of features, we need to re-train the neural network, and this takes time.
- It is thus desirable to have faster methods for finding the optimal value n_0.

5. Let Us Formulate the Problem in Precise Terms

- We are given a number N – the overall number of available features.
- We have an algorithm that:
  - given a natural number $n \leq N$,
  - returns a real number $E(n)$ – the effectiveness that we get if we consider n most promising features.
- We know that the function $E(n)$ first strictly increases, then strictly decreases.
- In other words, there exits some threshold value n_0 – that is not given to us – for which:
  - if $n < n' \leq n_0$, then $E(n) < E(n')$, and
  - if $n_0 \leq n < n'$, then $E(n) > E(n')$.
- We want to compute the threshold value n_0, i.e., the value at which the effectiveness $E(n)$ attains the largest possible value.

6. Usual method for computing n_0

- The usual method for computing n_0 is trying $n = 1$, $n = 2$, etc., until we reach the first value n for which $E(n) < E(n-1)$.
- Then, we return $n_0 = n - 1$.
- This method requires, in the worst case, N calls for the algorithm $E(n)$.
- We want to come up with a faster method for computing n_0.

7. New method: main idea

- For values $n < n_0$, we have $E(n) < E(n+1)$.
- For values $n \geq n_0$, we have $E(n) > E(n+1)$.
- It is therefore reasonable to use bisection to find the threshold value n_0.

8. Resulting method

- At each iteration, we have values $n_- < n_+$ for which
$$E(n_-) < E(n_- + 1) \text{ and } E(n_+) > E(n_+ + 1).$$
- Based on the properties described in the previous slide, this implies that $n_- < n_0 \le n_+$.
- We start with the values $n_- = 0$ and $n_+ = N - 1$.
- At each iteration, we take the midpoint $m = \left\lfloor \frac{n_- + n_+}{2} \right\rfloor$ and check whether $E(m) < E(m + 1)$. Then:
  - If $E(m) < E(m + 1)$, we replace n_- with the new value m.
  - If $E(m) > E(m + 1)$, we replace n_+ with the new value m.
- At each iteration, the width of the interval $[n_-, n_+]$ is decreased by half.
- We stop when this width becomes equal to 1, i.e., when $n_+ - n_- = 1$.
- Once we reach this stage, we return n_+ as the desired value n_0.

9. How many calls to the algorithm $E(n)$ this method requires

- We start with an interval $[0, N-1]$ of width $\approx N$.
- At each stage, the width of the interval decreases by a factor of 2.
- Thus, after k iterations, we get an interval of width $2^{-k} \cdot N$.
- The number k of iterations needed to reach the desired interval of width 1 can be therefore determined from the formulas $2^{-k} \cdot N = 1$.
- So $k = \log_2(N)$.
- On each iteration, we call the algorithm $E(n)$ twice: to find $E(m)$ and to find $E(m+1)$.
- Thus, overall, this method requires $2 \cdot \log_2(N)$ calls to the algorithm $E(n)$.

10. This method is asymptotically optimal

- We need to find a natural number n_0 from the interval $[0, N]$.
- In general, by using b bits, we can describe 2^b different situations.
- Thus, the amount of information b that we need to determine n_0 must satisfy the inequality $2^b \geq N$, i.e., equivalently, $b \geq \log_2(N)$.
- To get each bit of information, we need to call the algorithm $E(n)$.
- Thus, to find n_0, we need to make at least $\log_2(N)$ calls to this algorithm.
- The above algorithm requires $2 \cdot \log_2(N) = O(\log_2(N))$ calls.
- Thus, this method is indeed asymptotically optimal.

11. Natural question

- The straightforward method described in this section is asymptotically optimal, this is good.
- However, still, this method requires twice more calls to the algorithm $E(n)$ that the lower bound.
- Thus, a natural question is: can we make it faster?
- Our answer to this question is "yes".
- Let us describe the new faster method.

12. New Faster Method: Description

- *Preliminary stage:* First, we compute the value $E(m)$ for the midpoint m of the interval $[0, N]$.
- So we form an interval $[n_-, n_+] \stackrel{\text{def}}{=} [0, N]$.
- *Iterations:* At the beginning of each iteration, we have the values $n_- < n_+$ for which:
  - we know the values $E(n_-)$, $E(n_+)$ and $E(m)$, where m is the midpoint of the interval $[n_-, n_+]$, and
  - we know that $E(n_-) < E(m) > E(n_+)$.
- *When we stop:* when $n_+ - n_- = 2$, in which case we return $n_0 \stackrel{\text{def}}{=} n_- + 1$.
- *Iterations (cont-d):* at each iteration, we select, with equal probability 0.5, whether we start with the left or right subinterval.

13. If we start with the left subinterval

- We compute the midpoint L of this subinterval, and compute the value $E(L)$. Then:
- If $E(L) > E(m)$, i.e., if $E(n_-) < E(L) > E(m)$, then we replace the interval $[n_-, n_+]$ with the new half-size interval $[n_-, m]$.
- In this case, the iteration is finished.
- So, if the stopping criterion is not yet satisfied, we start the new iteration.
- On the other hand, if $E(L) < E(m)$, then we compute the midpoint R of the right subinterval $[m, n_+]$, and compute the value $E(R)$. Then:
  - If $E(m) > E(R)$, i.e., if $E(L) < E(m) > E(R)$, then we replace the interval $[n_-, n_+]$ with the new half-size interval $[L, R]$.
  - If $E(m) < E(R)$, i.e., if $E(m) < E(R) > E(n_+)$, then we replace $[n_-, n_+]$ with $[m, n_+]$.

14. If we start with the right subinterval

- We compute the midpoint R of this subinterval, and compute the value $E(R)$. Then:
- If $E(m) < E(R)$, i.e., if $E(m) < E(R) > E(n_+)$, then we replace the interval $[n_-, n_+]$ with the new half-size interval $[m, n_+]$.
- In this case, the iteration is finished.
- So, if the stopping criterion is not yet satisfied, we start the new iteration.
- On the other hand, if $E(m) > E(R)$, then we compute the midpoint L of the left subinterval $[n_-, m]$, and compute the value $E(L)$. Then:
  - If $E(L) > E(m)$, i.e., if $E(n_-) < E(L) > E(m)$, then we replace the interval $[n_-, n_+]$ with the new half-size interval $[n_-, m]$.
  - On the other hand, if $E(L) < E(m)$, i.e., if $E(L) < E(m) > E(R)$, then we replace the interval $[n_-, n_+]$ with $[L, R]$.

15. Why this algorithm works

- If for some values $n_- < m < n_+$, we have $E(n_-) < E(m) > E(n_+)$, then:
  - we cannot have $n_0 \leq n_-$, since then $n_- < m$ would imply $E(n_-) < E(m)$; thus, we must have $n_- \leq n_0$; and
  - we cannot have $n_+ \leq n_0$, since then $m < n_+$ would imply $E(m) < E(n_+)$; thus, we mist have $n_0 \leq n_+$.
- Thus, in this case, we must have $n_0 \in [n_-, n_+]$.

16. How many calls to the algorithm $E(n)$ this method requires

- Each iteration reduced the width of the original interval $[n_-, n_+] = [0, N]$ by half.
- So, similarly to the straightforward algorithm, we need $\log_2(N)$ iterations
- On each iteration, we first make the first call to $E(n)$ and then the first comparison.
- We have two possible results of this comparison, so it is reasonable to assume that each comparison result occurs with probability 0.5.
- So, on each iteration:
  - with probability 0.5, we require only one call to the algorithm $E(n)$, and
  - with probability 0.5, we require two calls.
- Thus, the expected number of calls on each iteration is

$$0.5 \cdot 1 + 0.5 \cdot 2 = 1.5.$$

17. How many calls to the algorithm $E(n)$ this method requires (cont-d)

- We make an independent random selection on each iteration.
- Thus, the numbers of calls on different iterations are independent random variables.
- So, due to the large numbers theorem, the overall number of calls will be close to the expected number of calls, i.e., to $1.5 \cdot \log_2(N)$.
- This is clearly faster than the number of calls $2 \cdot \log_2(N)$ required for the straightforward algorithms.

18. Conclusions

- In training a neural network, it is important to select the proper number of features.
- If we select too few features, we may lose important information, and thus, decrease the prediction accuracy and effectiveness.
- On the other hand:
  - if we include too many features, including ones that barely affect the predicted quantity,
  - these barely-affecting features act, in effect, as noise, and the effectiveness of prediction decreases again.

19. Conclusions (cont-d)

- At present, to find the optimal number of features, practitioners:
  - sort them in the order of their prospective usefulness,
  - e.g., by the absolute value of the correlation between the feature and the desired value, and
  - then add features one by one in this order until we reach the largest possible effectiveness (and the smallest prediction errors).
- The main limitation of this process is that:
  - for each added feature, we need to re-train the neural network, and
  - each re-training takes a significant amount of time.
- It is therefore desirable to be able to decrease the number of re-trainings needed to reach the most effective number of features.

20. Conclusions (cont-d)

- In this talk, we present a method that drastically decreases the number of such re-trainings in comparison to the usual practice.
- Moreover, we show that this method is (asymptotically) optimal, i.e., it leads to the (asymptotically) smallest number of re-trainings.

21. Acknowledgments

This work was supported in part by:

- National Science Foundation grants 1623190, HRD-1834620, HRD-2034030, and EAR-2225395;
- AT&T Fellowship in Information Technology;
- program of the development of the Scientific-Educational Mathematical Center of Volga Federal District No. 075-02-2020-1478, and
- a grant from the Hungarian National Research, Development and Innovation Office (NRDI).