Optimization of Feature Selection

James (Wonshik) Choi Esteban G. Tabak

September 2019

Abstract

A methodology is developed to optimize the feature selection in the process of predicting a quantity of interest using a selection of features. It provides indices of the features, w's, that can be used to minimize the error between the prediction and the true value of x, a quantity of interest.

1 Introduction

Given a sample consisting of a quantity of interest, $\{x^i\}$, with its features / attributes $\{w_h^i\} = \{w_1^i, w_2^i, \cdots\}$, many machine learning algorithms rely on studying the i sample points and finding a relationship between the x's and the w's. The algorithms, then can use the found relationship to predict the quantity x^n given the attributes $\{w_h^n\}$.

Now suppose instead of using all the attributes to make the prediction, one wishes to use only a proportion of them to make the prediction. There could be various reasons for wanting to limit the number of features. For example, a doctor may know that to lower his patient's blood pressure, there are 60 different things that can be done. However, instead of bombarding his patient with too much information, the doctor might realize that the sodium intake and amount of exercise are the two most important factors, and tell his patient to focus on those two.

This paper proposes one method to solve this feature selection problem. Section 2 describes how to construct a model of prediction, using the idea of low rank factorization, to find the relationship between the quantity of interest and the features. Section 3 describes how a number of features can be reduced by introducing new features $\{z_l^i\}$, which are convex combinations of the original features $\{w_h^i\}$. Also described in Section 3, is the use of penalty function in order to turn each of these convex combinations into a singular term. In other words, each z would equal exactly one of the w's.

In each step, some form of optimization is involved, in order to reduce the gap between the real values of x's and the prediction made by z's and w's.

Unfortunately, the nature of the optimization process sometimes causes the algorithm to be stuck at a local minimum. To get around this problem, a random restart is used, which is discussed in further details in section 4.

2 Model of Prediction

2.1 Constructing the Model of Prediction

To predict the quantify of interest x using the attributes z's, we first need to go through the sample points and establish the relationship between the $\{x^i\}$ and the $\{z_l^i\}$. In other words, we need to find the function F such that $x^n \approx F(\{z_l^n\})$ for each n. We claim that this F can be approximated by sums and products of simpler functions that each takes only one of the z's.

$$x \approx \sum_k \prod_l F_l^k(z_l) \tag{1}$$

The two motivations for the claim comes from low rank factorization of matrices and separation of variables in two dimensional heat equation.

For discrete $\{z_l\}$, we can apply the result from linear algebra that for any matrix A, we can write $A = \sum_{k=1}^{r} \sigma_k u_k v_k^T$, where $r = rank(A)$, σ_k are the singular values of A, and u_k, v_k are some column vectors. The low rank factorization of A would be $A \approx \sum_{k=1}^{n} \sigma_k u_k v_k^T$ for some $n < r$.

By letting $\bar{u}_k = \sigma_k u_k$, we can rewrite this equation as $A \approx \sum_{k=1}^{n} \bar{u}_k v_k^T$, or $A_{ij} \approx \sum_{k=1}^{n} \bar{u}(i)v(j)$, which is equivalent to $A(i,j) \approx \sum_{k=1}^{n} \bar{u}(i)v(j)$. Extending this to tensors, we can write $A(z_1, z_2, \cdots, z_m) \approx \sum_{k=1}^{n} F_1^k(z_1)F_2^k(z_2)\cdots F_m^K(z_m) = \sum_{k=1}^{n}\prod_{l=1}^{m} F_l^k(z_l)$, which is equivalent to (1).

For continuous $\{z_l\}$, we take the motivation from the fact that for $T(x,y,t)$, the temperature at point (x,y) at time t, we can write
$T(x,y,t) = \sum_m \sum_n a_{mn} sin(\mu_m x) cos(\nu_n y) e^{-\lambda_{mn}^2 t}$. This is an example of a function of multiple variables following the form of (1) even when the variables are not discrete.

Now for any function F_l^k, we should be able to approximate it using a basis of simpler functions. i.e. $F_l(z_l) \approx \sum_j a_j^{kl}\phi_j(z_l)$, where ϕ_j's are simpler functions such as polynomials. Therefore,

$$x \approx \sum_k \prod_l \sum_j a_j^{kl}\phi_j(z_l) \tag{2}$$

Now suppose that for some n, there is no correlation between x and z_n. Then, it seems appropriate that F_n^k should equal to some nonzero constant for all k.

For this to be true, as $F_n^k \approx \sum_j a_j^{kn}\phi_j$, we should make our approximation with some ϕ_j equal to a constant function.

$$x \approx \sum_k \prod_l [a^{kl} + \sum_j a_j^{kl}\phi_j(z_l)] \tag{3}$$

However, the constants a^{kl}'s can cause gauge invariance, and result in fluctuations of a_j^{kl}'s. Therefore, instead of using $k \cdot l$ different variables a^{kl}'s, we will be replacing them by 1, and using a global constant c to match our prediction with the quantity of interest in our samples.

Now suppose we were to multiply our prediction by c, so that $x \approx c\sum_k \prod_l [1 + \sum_j a_j^{kl}\phi_j(z_l)]$. The problem with this model can be seen in an example where $x = z_1$, k = 1 and $\phi_1(z) = z$. In this case, all the F_l^k would set to be 1 for $l \neq 1$. And we would have $x \approx c(1 + az_1)$. In this case, because of the constant 1, c would bet smaller and smaller and a would get bigger and bigger as we try to minimize the error. For our method for solving for a_j^{kl} which will be discussed in section 2.2, the a's getting too big can become problematic. Therefore, instead of multiplying, we add a constant c to our prediction, and our model of prediction is

$$x \approx c + \sum_k \prod_l [1 + \sum_j a_j^{kl}\phi_j(z_l)] \tag{4}$$

2.2 Solving for the a's (Alternating Method)

A paper by Esteban Tabak and Giulio Trigila suggests that an optimization of a system in the form of (1) can be solved by solving for each F_l^k, one l at a time. In other words, we shall solve for F_1^k with all the other F_l^k with $l \neq 1$ fixed, then solve for F_2^k, and carry on in a cycle. In this paper, this method of optimization will be called an *Alternating Method*, as we alternate the function variable which we are solving for.

With our model of prediction (4), this is equivalent to solving for the a_j^{kl}, for each l at a time. Also, every time we solve for a_j^{kl}, it seems appropriate that we update c as well. Without changing c, the mean of $\sum_k \prod_l [1 + \sum_j a_j^{kl}\phi_j(z_l)]$ would stay at a constant value, $\bar{x} - c$, which is a constraint that is unnecessary.

For any m, We can rewrite (4) as

$$\begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^n \end{bmatrix} \approx \begin{bmatrix} B_1^m & B_2^m & \cdots & B_q^m & J_{n,1} \end{bmatrix} \begin{bmatrix} a_1^{1m} \\ a_2^{1m} \\ \vdots \\ a_p^{1m} \\ a_1^{2m} \\ \vdots \\ a_p^{qm} \\ c \end{bmatrix} \tag{5}$$

where n is the number of sample points, p is the number of ϕ's used to approximate F_l^k's, and q is the number of rank used in the model of prediction. $J_{n,1}$ is defined to be the n x 1 matrix of ones and B_k^l is defined so that

$$B_k^l = \prod_{l' \neq l} F_{l'}^k(z_{l'}) \begin{bmatrix} \phi_1(z_l^1) & \cdots & \phi_p(z_l^1) \\ \vdots & \ddots & \vdots \\ \phi_1(z_l^n) & \cdots & \phi_p(z_l^n) \end{bmatrix} \tag{6}$$

Writing (5) as $x \approx Ba_m$, we can find $\bar{a}_m = (B^T B)^{-1} B^T x$ such that $\bar{a}_m$ is the choice of a_m that minimizes the 2-norm of $x - Ba_m$.

The alternating method involves replacing a_j^{kl} by $\bar{a}_l$ for each l until the a's converge. The correctness of our model of prediction and the alternating method is demonstrated in Figure 1. The third plot of Figure 1 shows that a prediction with full rank produces the exact match

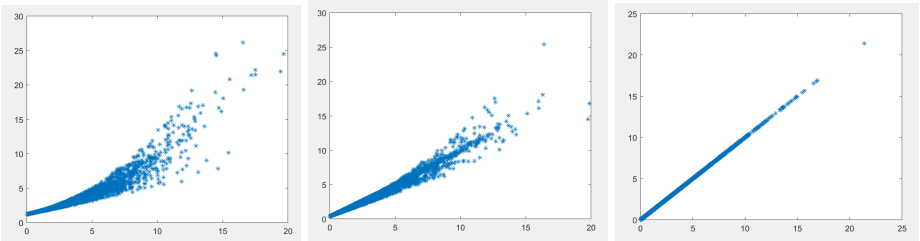

Figure 1: Plots of the actual x values vs our prediction of x using the model and method described in this paper. The quantity of interest had rank 3, and the prediction was made with rank 1, rank 2, and rank 3 respectively.

3 Reducing the Number of Attributes

3.1 Introducing the γ's

Now that we have our model of prediction, our next task is to choose our z's out of the given w's so that the best prediction can be made. An obvious solution

to this problem would be to try different combinations of w's and compare the results to find the best combination. However, this becomes very costly as the number of w gets larger.

Instead, we let z's be convex combinations of w's. In other words, $z_l = \sum_h \alpha_h^l w_h$, where each α's are nonnegative, and $\sum_h \alpha_h^l = 1$ for each l. To enforce these constraints, we introduce new variables γ's, and let $\alpha_h^l = \frac{{\gamma_h^l}^2}{\sum_m {\gamma_m^l}^2}$. So we can rewrite our model of prediction as

$$\begin{aligned} x &\approx c + \sum_k \prod_l [1 + \sum_j a_j^{kl} \phi_j(z^l)] \\ &= c + \sum_k \prod_l [1 + \sum_j a_j^{kl} \phi_j(\sum_h \alpha_h^l w^h)] \\ &= c + \sum_k \prod_l [1 + \sum_j a_j^{kl} \phi_j(\sum_h \frac{{\gamma_h^l}^2}{\sum_m {\gamma_m^l}^2} w_i^h)] \end{aligned} \tag{7}$$

To find the right values of γ's, we'll be using the gradient descent method. We still wish to minimize the 2-norm of the error between the x's and our predicted x's. Therefore, the objective function we aim to minimize would be

$$obj = \sum_i [x_i - c - \sum_k \prod_l [1 + \sum_j a_j^{kl} \phi_j(\sum_h \frac{{\gamma_h^l}^2}{\sum_m {\gamma_m^l}^2} w_i^h)]]^2 \tag{8}$$

Taking the derivative of this in respect to γ_h^l, we get

$$\frac{\delta}{\delta \gamma_h^l} obj = -2 \sum_i error_i \sum_k \prod_{L \neq l} F_L^k(z_i^L) \sum_j a_j^k l \phi_j'(z_i^l) \frac{2\gamma_h^l}{S_l} (w_i^h - \sum_H \frac{{\gamma_H^l}^2}{S_l} w_i^H) \tag{9}$$

where $error_i = x_i - c - \sum_k \prod_l [1 + \sum_j a_j^{kl} \phi_j(z_i^l)]$, $F_l^k(z_i^l) = 1 + \sum_j a_j^{kl} \phi_j(z_i^l)$, ϕ_j' is the derivative of ϕ_j, and $S_l = \sum_h {\gamma_h^l}^2$.

Just like solving for a's, we solve for γ's for each l separately. At each iteration, we move the γ's in the direction of the negative of the gradient. Hence, the new set of γ's, γ', is defined by reducing the components of γ corresponding to l by $\beta \frac{\delta}{\delta {\gamma_h^l}^2} obj$ where β is a step size chosen arbitrarily. And then, if the objective function value with γ' is higher than that with the original γ, we reduce β until the new objective function value is lower than the original.

Repeating this procedure along with the alternating method described above, we solve for a's and γ's simultaneously. The order of our algorithm is: solve for $a_j^{k1} \rightarrow$ solve for $\gamma_h^1 \rightarrow$ solve for $a_j^{k1} \rightarrow$ solve for $a_j^{k2} \rightarrow$ solve for $\gamma_h^2 \rightarrow$ solve for $a_j^{k2} \rightarrow \cdots$

3.2 Using the Penalty Function

So far, we've reduced the number of attributes by substituting the w's by z's. However, this is not good enough as the z's are still convex combinations of the w's, i.e. $z_l = \sum_h \alpha_h^l w_h$. To achieve our goal, we need each z to equal a distinct w, which means that for each l, α_h^l must equal 1 for exactly one of the h's and zero for the others. In other words, for each l, γ_h^l must equal 0 for all the h's except for exactly one of them. In this paper, this will be called the *Separation of the Gammas*.

As the γ's are modified during the gradient descent process, we need to make sure that the separation of the gammas happen during gradient descent. To do this, we modify our objective function and add a penalty function, which will decrease in value as the separation of the gammas happen.

$$obj = \sum_i error_i^2 + \mu \cdot penalty \tag{10}$$

Here, μ is the penalty constant which determines how much the objective function is penalized for not having its γ's separated. There are many options for the penalty function; one simple choice would be $-\sum_{h,l} {\alpha_h^l}^2$. By doing the gradient descent with an appropriate value of μ, the γ's will be solved so that the error is minimized and the separation of the gammas is achieved simultaneously.

3.3 Choice of μ

For the gradient descent method to work with the new objective function (9), the choice of μ is important. If μ is too big, the separation of the gammas would happen way too quickly and the algorithm would be stuck at a local minimum. This is useless as it is equivalent to selecting random z's out of the w's without considering the minimization of the error. On the other hand, if μ is too small, the gradient descent would prioritize minimizing the error that the separation of the gammas may never happen.

In fact, it seems almost impossible to find the right value of μ. Although the optimal penalty value is known ($-l$), the optimal error value is unknown, which makes it difficult to find a constant that will balance out the two. Instead, it seems more appropriate that μ changes over time. More precisely, it seems logical to have μ small in the beginning and have it get bigger gradually. This way, in the beginning of the program, the algorithm looks to minimize the error without worrying too much about the separation of the gammas. Then, once the error is small and the program is not making much progress, μ should get bigger and enforce the separation of the gammas.

Considering this, a good choice of μ would be $\frac{1}{y_i - \bar{y}_i}$, where y_i is the prediction made with (7) with the a and γ values before the current gradient descent iteration, and $\bar{y}_i$ is the prediction made with the a and γ values a step (one

iteration of alternating method) before that. However, this is not good enough. Since the alternating method decreases the error value but may increase the objective function, and the gradient descent decreases the objective function but may increase the error value, sometimes the program may end up in a loop where the y_i values fluctuate. In this case, the μ value will not get larger and the program may never converge. To prevent this, we ensure that μ always gets larger, even if at a very slow rate.

$$\mu_i = max(\mu_{i-1}, \frac{1}{|y_i - \bar{y_i}|}) \tag{11}$$

4 Stopping and Restarting

Due to the nature of the optimization process, our algorithm can get stuck at a local minimum. There seems to be no easy way to get around this problem. The best we can do is to restart the program at a random point, and try this multiple times until we find the global minimum.

4.1 Stopping the Program

To perform random restart, first we need to know when to stop the program. A good place would be when enough separation of the gammas has happened, as once the gammas have been separated, it is unlikely that the choice of z's would be altered. A good way to measure the separation of the gammas is by using the penalty function.

Let $\bar{\alpha}_l$ denote the maximal α_h^l corresponding to l. The optimal case would be when all the $\bar{\alpha}_l$'s are at 1, which means the separation of the gammas is complete. In this case, the value of the penalty function would be $-l$. However, stopping the program when $penalty = -l + 0.3$ should be sufficient. Even at the worst case, where all but one $\bar{\alpha}_l$ are at 1, calculations show that $\min_l \bar{\alpha}_l$ would be at around 0.82. This is sufficient for the decision of z's.

Now with the stop condition, we have our full algorithm for one try of finding the z's. The pseudocode is provided in Algorithm 1, TrySolve. TrySolve takes x, w, and random values of a's and γ's. Then, it returns the indices of w's selected for the final z's and the values of a's that were used with those z's to minimize the error.

Algorithm 1 TrySolve

$h \leftarrow$ the number of total attributes
$i \leftarrow$ the number of sample points
$j \leftarrow$ the number of ϕ's used
$k \leftarrow$ the rank of approximation
$l \leftarrow$ the number of attributes selecting
$\bar{y} \leftarrow i$ x 1 vector of zeros
$y \leftarrow i$ x 1 vector of ∞
$z \leftarrow$ created using w's and the given γ values
while $l + penalty > 0.3$ **do**
  for $var = 1 : l$ **do**
    $\bar{y} \leftarrow y$
    update a_j^{kvar} by using Alternating Method
    update y
    $\mu \leftarrow max(\mu, \frac{1}{|y-\bar{y}|})$
    update γ_h^{var} by using Gradient Descent
    update z's using the new γ's
    update a_j^{kvar} by using Alternating Method
  end for
end while
return a, and array of h's such that $\alpha_h^l > 0.5$ for some l

4.2 Random Restart

When TrySolve algorithm returns an array of of indices, these indices may not be valid. This is the case, when there are repeated indices. Clearly, this is not optimal, however there still can be some information that can be retrieved from this. If an attribute w_m is selected twice or more, it is likely that w_m is closely related to the quantity of interest. Therefore, although a random restart is needed for this case, we set α_m^1 to be 1 for the next TrySolve iteration.

On the other hand, if the indices are valid, we use the given indices and the a's to run Alternating Method to find the error value associated with these indices. We record the indices and the error value associated with these indices. Then we run another iteration of TrySolve with completely random gammas. We repeat this process and continue to update the minimum error value and the indices corresponding to the minimum error. The program ends when the same minimum error value is obtained λ times (The higher the value of λ is, the slower and more accurate the program is). The psuedocode for the full algorithm is given in Algorithm 2.

The ϵ value is included in the algorithm due to the fact that the program doesn't always return the minimum error value even if the optimal indices are used.

Algorithm 2 FullSolve

```
count ← 0
minError ← ∞
minAttr ← lx1 vector of zeros
λ ← the number of minimum error wanted
while count > λ do
  get a and attr by running TrySolve
  reset γ's to random
  if index m is repeated in attr then
    γ_m^1 ← 1
    for l ≠ 1 do
      γ_m^l ← 0
    end for
    for h ≠ m do
      γ_h^1 ← 0
    end for
  else
    errorNorm ← the error value computed using the indices from attr with
    the Alternating Method
    if errorNorm < minError + ε then
      if attr = minAttr or |minError − errorNorm| < ε then
        count ← count + 1
        if errorNorm < minError then
          minError ← errorNorm
          minAttr ← attr
        end if
      else
        minError ← errorNorm
        minAttr ← attr
        count ← 0
      end if
    end if
  end if
end while
return minAttr
```

5 Conclusion

This paper describes a two-stage method which finds the optimal features to be used when a selection of features is to be used for a model of prediction of quantity of interest. The first stage involves using the alternating method and gradient descent simultaneously in order to find the features which minimizes the error function locally. Then, these local minimum values is evaluated multiple times at the second stage to find the absolute minimum value.

By the nature of an algorithm that uses random restart, the accuracy and the speed of the algorithm can vary depending on the values of variables such as λ and ϵ. Optimization of these values could be further looked into. Testing the algorithm with a real data is also something that the authors intend to do in the near future.