An interactive application for Regularized Linear Models

Michiel Ruelens
Promoters Olivier Gevaert Roel Wuyts

Abstract— **This document explains the development of an interactive application for constructing and evaluating regularized linear models. The models themselves are computed using the glmnet package [1] for the R scripting language. The application is built using the shiny framework for R which provides a link between R-scripts and a front-end website.**

I. INTRODUCTION

Generalized linear models are known for a long time in the fields of machine learning and data analysis. But the addition of regularization over the last few years has really made these models much more powerful. Combine this with a really fast implementation such as provided by glmnet [2] and these models really become state-of-the-art. In this paper I will explain why the addition of regularization is such a big step and how an interactive application can allow non-experts to use and interpret these regularized linear models.

II. BACKGROUND

A. Generalized Linear Models

The basis for generalized linear models are standard linear models. These models simply make a linear combination (called weights) of the input variables to produce an output. Mathematically this can be written as:

$$s = \sum_{i=1}^{d} w_i x_i$$

Where s is the output variable, x_i is the i'th input variable, and w_i is the weight corresponding to x_i. One of the simplest methods one could use is called binary linear classification. In this case we would pick a threshold. If the output s is greater than the threshold we predict one class, otherwise we predict the other class. This however is one of many possible things one could do. We could directly use the numeric value of the output as prediction, in this case the method is called linear regression. If the output is sent to a sigmoid function, we could treat it as a probability and we would call the method logistic regression.
The only difference in all these methods is the kind of output variable we are trying to predict: classes, numeric values, probabilities, ... the routine for finding the output variable however is the same for all. Therefore all these methods have been grouped under a common name: generalized linear models (GLM).

B. Computing a GLM

The way a generalized linear model is computed is by using a very well known technique called gradient descent. In this method, we start out with an initial set of weights and every iteration we improve these weights by a small amount. Repeat this process a sufficient amount of times and the weights will converge to a solution. This is possible because the weights are updated by using a so called error-surface. We define an error between our current solution and the target. We then update our weights in order to minimize this error, and therefore bring our solution closer to our target. One can view that process also as finding the minimum of the error-surface. Since the error-surface in the case of linear models is a convex surface, one can prove this method will always converge to a minimum [3].

C. Problems with GLM

A generalized linear model, computed using gradient descent, will provide a solution that can be used to predict outcomes for a set of input variables. However there is a problem with these kind of methods. The problem is often called overfitting [5]. Overfitting means that the model we have found is very good at predicting the outcomes for cases that appeared in the training set (the set of inputs that were used to find the model in the first place using gradient descent). However the model is not very good at predicting outcomes for cases outside of the training set. But this is exactly what we want the model to do. Overfitting often happens when the model we choose is too complex. Complexity, in the case of linear models, can be very roughly measured by the amount of input variables, because we associate one weight with each input variable. In current real life problems however we have a huge amount of input variables, and thus a huge amount of weights making the model very complex. These very complex models are able to fit the training data so well that overfitting makes the model completely useless outside of the training set.

D. Bias-variance tradeoff

The last things to go over, before talking about regularization, are the concepts of bias and variance [5]. In the general machine learning case we are trying to find a model h to fit a target function f. We choose a certain set of models (e.g. linear models) from which we are going to pick h. We call this set the hypothesis set. It is very important to realize that this is a choice we make, and this is a restriction we are putting on ourselves. We are confident that there is a hypothesis h in the set that fits the target function f well.

However because we are limiting ourselves to this specific hypothesis set, we are possibly making an error. We can never be sure that there even exists a hypothesis that fits the target perfectly. Call $\overline{h}$ the best possible hypothesis in our hypothesis set. This means that $\overline{h}$ is the hypothesis that best fits our target f. However, we know that $\overline{h}$ might not be perfectly able to fit f. We call this error the bias. It is an error we make due to the fact that we chose to use this specific hypothesis set.
On the other hand, when we actually use a machine learning technique to find a model h in our hypothesis set, we won't always find the best one ($\overline{h}$). If we call the model that we actually find h. Then we can define the error between $\overline{h}$ and h as the variance. This is an error we make due to the fact that our machine learning method is not perfect, and as such doesn't always find the optimal solution. Notice that the variance is linked to the size of the hypothesis set we choose. A larger hypothesis set can have very large variances, a very small hypothesis set is bound to have a small variance. This brings us directly to the concept of regularization.

III. Regularization

The ending of the last section immediately raises the question: is there a way we can influence bias and variance in order to force our machine learning method to find better models (reduce the overall error). This is exactly what regularization does. It will introduce additional bias (which raises the error), in order to reduce the variance significantly. More precisely it will add additional constraints to our hypothesis set. There are three main constraints that are used nowadays and they are called ridge-, lasso- and elastic net regularization.

A. *Regularization types*

In ridge regularization the added constraint is a maximum on the sum of squared weights.

$$\sum_{i=1}^{d} w_i^2 \leq C$$

Where C is the constraint constant that defines how strong the constraint is. The result of this constraint is that the model tends to go for smaller weights, and only if the weight is really important will they get larger. This form of regularization is also called the L_2 penalty.
In lasso regularization the added constraint is a maximum on the sum of the absolute values of the weights.

$$\sum_{i=1}^{d} |w_i| \leq C$$

Again, C is the constraint constant that defines how strong the constraint is. In this case the model doesn't only tend to prefer smaller weights, but this form of regularization also performs variable selection. This means that it will tend to make certain weights zero if they are not considered important. This is especially useful nowadays because we often have very large amounts of input variables, and we want to use only these variables that are actually useful. It also reduces the size of the model we end up with. This form of regularization is also called the L_1 penalty.
The last form of regularization I will discuss is the elastic net. This type is really just a combination of the two previously mentioned L_1 and L_2 penalties.

$$\alpha \sum_{i=1}^{d} w_i^2 + (1-\alpha) \sum_{i=1}^{d} |w_i| \leq C$$

Where α indicates the proportion of L_1 and L_2 penalty. This type of regularization has the added benefit that it performs well in the case of group selection. Sometimes there are sets of input variables that are very correlated. In this case we would like to have the corresponding weights correlated too. The elastic net penalty tends to do this properly, while the other two penalties do not [4].

There are a number of reasons why regularization helps us to find better models. It tends to prefer smaller weights, making it impossible for a small number of weights to get out of control. It performs automatic variable selection reducing the model size and helping eliminate useless input variables. It also allows for group selection which can be very useful in certain scenarios. And lastly but most importantly it strongly reduces the possibility of overfitting by the introduction of the constraints. There will be a slight bias error introduced, but the amount we gain by lowering the variance error is far more important.

IV. Evaluating a model

A. *Cross validation*

To evaluate a model we would like to use the model on some examples that it hasn't seen before (that haven't been used for training). However what we usually have is one set of examples and we also would like to use all of them to train our model. The solution here is cross-validation. In this technique we split up the initial dataset into chunks (usually ten). We then proceed to use one of the chunks as test set and use the other nine to train a model. We repeat this process ten times, using each chunk once as the test set. If we then take the average performance of the ten models we just computed we can get an estimate for the performance of the model that we would get if we used the full set for training.

B. *ROC analysis*

The way in which we test the performance of a single model against a test set is called ROC analysis. In this method, we distinguish four different cases when a model predicts an outcome for a test example.

- True positive: the model predicts positive, and the true condition is positive
- True negative: the model predicts negative, and the true condition is negative
- False positive: the model predicts positive, and the true condition is negative

- False negative: the model predicts negative, and the true condition is positive

These four cases reflect the performance of the model. In practice however we usually don't use these numbers by themselves, but rather combine them to make new characteristics for the model. The five main characteristics used are accuracy, sensitivity, specificity, positive predictive value (PPV) and negative predictive value (NPV).

- $Accuracy = (TP + TN)/(TP + TN + FP + FN)$
- $Sensitivity = TP/(TP + FN)$
- $Specificity = TN/(TN + FP)$
- $PPV = TP/(TP + FP)$
- $NPV = TN/(TN + FN)$

A last metric that is often very indicative of the model performance is the Area Under the Curve (AUC). This is the area under the ROC curve that is obtained by plotting sensitivity against (1-specificity) for all possible thresholds, or also called operating points. The area under the curve is an indication of how well the model is able to distinguish between positive and negative cases. A model that randomly predicts positive and negative would have an AUC of 0.5. The perfect model would have an AUC of 1. In practice models that have an AUC below 0.7 are considered bad. Between 0.7 and 0.9 is acceptable. Higher than that would be exceptional.

In all of this there is really only one degree of freedom. It is the threshold that determines whether a certain output is classified as a positive or a negative case. Changing this threshold will change the predicted outcome for the examples and thus influence all the aforementioned metrics.

V. Interactive Application

If someone wants to use regularized linear models and evaluate their performance, the details of the algorithms can get very tedious. It is for this very reason that I will develop an interactive application using the shiny framework for R. The application will make abstraction of all the details of regularized linear methods and simply provide the end-user with sliders to control the variables that really matter. One of these variables is the threshold or operating point. This point directly influences the evaluation metrics discussed earlier and is very application dependent. Some applications will require a very high sensitivity, while others really don't care about this but need a very high specificity or PPV. Changing the operating point can allow end-users to search for the optimal point for their specific application.

In current applications their are often multiple data sources. It is not always clear in advance which data sources should be used and how they should be combined. For instance the data sources could all be combined at the very start to create 1 big data source and use that for learning (called early integration). Another approach could be to treat each source individually and construct a model for each data source. Then when predicting the outcome for a test example, one could take the outcomes of all the models and combine them to create a final answer (called late integration). The application could perform different kind of integration methods at the same time and provide the user only with the best method, or allow the user to specify the kind of integration it wants to use.

The application would thus deal with the tedious bits so the end-user can focus on interpreting the results and building the optimal model for their purpose.

References

[1] https://cran.r-project.org/web/packages/glmnet/index.html

[2] Jerome Friedman, Trevor Hastie, Rob Tibshirani, Regularization Paths for Generalized Linear Models via Coordinate Descent, Journal of Statistical Software January 2010, Volume 33, Issue 1

[3] Geoffrey Hinton; Nitish Srivastava; Kevin Swersky. Part of a lecture series for the Coursera online course Neural Networks for Machine Learning

[4] Hui Zou and Trevor Hastie, Regularization and variable selection via the elastic net. Stanford University, USA, 2004

[5] Yaser Abu-Mostafa, Machine Learning online course, http://work.caltech.edu/telecourse.html