Feature Selection, Sparsity, Regression Regularization

1 Feature Selection Introduction

`from Wikipedia`

A feature selection algorithm can be seen as the combination of a search technique for proposing new feature subsets, along with an evaluation measure which scores the different feature subsets. The simplest algorithm is to test each possible subset of features finding the one which minimizes the error rate. This is an exhaustive search of the space, and is computationally intractable for all but the smallest of feature sets. The choice of evaluation metric heavily influences the algorithm, and it is these evaluation metrics which distinguish between the three main categories of feature selection algorithms: wrappers, filters and embedded methods.

Task Independent, Model independent. Assess correlation/dependency between feature irrespective of the task and the predictive model. See PCA.

Filter methods (Task dependent, Model independent) use a proxy measure instead of the error rate to score a feature subset. This measure is chosen to be fast to compute, whilst still capturing the usefulness of the feature set. Common measures include the mutual information, the pointwise mutual information, Pearson product-moment correlation coefficient, inter/intra class distance or the scores of significance tests for each class/feature combinations. Filters are usually less computationally intensive than wrappers, but they produce a feature set which is not tuned to a specific type of predictive model. Many filters provide a feature ranking rather than an explicit best feature subset, and the cut off point in the ranking is chosen via cross-validation. Filter methods have also been used as a preprocessing step for wrapper methods, allowing a wrapper to be used on larger problems.

Wrapper methods (Task dependent, Model dependent) use a predictive model to score feature subsets. Each new subset is used to train a model, which is tested on a hold-out set. Counting the number of mistakes made on that hold-out set (the error rate of the model) gives the score for that subset. As wrapper methods train a new model for each subset, they are very computationally intensive, but usually provide the best performing feature set for that particular type of model.

Forward selection (Task dependent, Model dependent) starts with

no feature(variable) in the model, and adds features to the model one at a time. At each step, the feature that can contribute most to the model is added. The procedure is repeated until one new feature can improve the model significantly(defined by some statistical test threshold).

set of features initial empty, $S = \emptyset$
repeat while improvement $> \epsilon$
    for each feature $f \notin S$
        performance $(S \cup \{f\})$ = performance(model, trained on $S \cup \{f\}$)
    end for
    f_{new} = argmax$_f$ performance($S \cup \{f\}$)
    improvement = performance $(S \cup \{f_{new}\})$ - performance (S)
    $S = S \cup \{f_{new}\}$
end repeat

Backward selection in contrast to Forward Selection, starts with all feature(variable) in the model, and eliminates features from the model one at a time. At each step, the feature that can contribute least to the model is eliminated. The procedure is repeated while there is no decrease in performance.

Embedded methods(Task dependent, blended with Model) are a catch-all group of techniques which perform feature selection as part of the model construction process. The exemplar of this approach is the LASSO method for constructing a linear model, which penalizes the regression coefficients, shrinking many of them to zero. Any features which have non-zero regression coefficients are "selected" by the LASSO algorithm. One other popular approach is the Recursive Feature Elimination algorithm, commonly used with Support Vector Machines to repeatedly construct a model and remove features with low weights. These approaches tend to be between filters and wrappers in terms of computational complexity.

In statistics, the most popular form of feature selection is stepwise regression. It is a greedy algorithm that adds the best feature (or deletes the worst feature) at each round. The main control issue is deciding when to stop the algorithm. In machine learning, this is typically done by cross-validation. In statistics, some criteria are optimized. This leads to the inherent problem of nesting. More robust methods have been explored, such as branch and bound and piecewise linear network.

Many popular search approaches use greedy hill climbing, which iteratively evaluates a candidate subset of features, then modifies the subset and evaluates if the new subset is an improvement over the old. Evaluation of the subsets requires a scoring metric that grades a subset of features. Exhaustive search is generally impractical, so at some implementor (or operator) defined stopping point, the subset of features with the highest score discovered up to that point is selected as the satisfactory feature subset. The stopping criterion varies by algorithm; possible criteria include: a subset score exceeds a threshold, a program's maximum allowed run time has been surpassed, etc.

Alternative search-based techniques are based on targeted projection pursuit which finds low-dimensional projections of the data that score highly: the features that have the largest projections in the lower-dimensional space are then selected. For a complete survey of feature selection methods, see [3].

2 The "Bet on Sparsity" Principle

book: Elements, 16.2.2

book: Statistics for high dimensional data, introduction

$$Y_i = \mu + \sum_{j=1}^{d} w_j X_i^j + \epsilon_i$$

Roughly speaking, for High-dimensional statistical inference to achieve reasonable accuracy or asymptotic consistency, we need

$$\log(d)(\text{sparsity}(w)) << n$$

3 Regularized Linear Models

3.1 Regularized Linear Regression

Consider the linear regression model with parameters $w = (w^1, w^2, ..., w^d)$

$$h_w(x) = w^0 + x^T w = w^0 + \sum_j x^j w^j$$

Suppose we have N data points, and d features. Each feature is standardized to have mean 0 an variance 1. Regularized Linear Regression solves the following problem:

$$\min_{(w_0,w)\in\mathcal{R}^{d+1}} \big[\frac{1}{2N}\sum_{i=1}^{N}(y_i - w_0 - x_i^T w)^2 + \frac{\lambda}{N}R(w)\big]$$

where $\sum_{i=1}^{N}(y_i - w_0 - x_i^T w)^2$ is the square loss term, $R(w)$ is a penalty term due to complexity of the model, and λ controls the strength of the R penalty:

- small λ means that complexity penalty is negligible, so the optimization is essentially solving a simple regression problem without regularization.
- large λ means that complexity penalty overwhelms the error, thus leading to small/low complexity w but large square error

There are three kinds of commonly used penalties in linear regression:

- an L_2 norm $R(w) = \frac{1}{2}||w||_2^2 = \frac{1}{2}\sum_{j=1}^{d} w^{j2}$ is called the RIDGE-regression penalty.

- an L_1 norm $R(w) = ||w||_1 = \sum_{j=1}^{d} |w^j|$ is called the LASSO penalty.
- a mixture $R_\beta(w) = (1-\beta)\frac{1}{2}||w||_2^2 + \beta||w||_1 = \sum_{j=1}^{d}[\frac{1}{2}(1-\beta)w^{j2} + \beta|w^j|]$ is called the elastic-net penalty. The elastic-net penalty is a compromise between the ridge-regression penalty and the lasso penalty.

Ridge Regression shrinks the size the regression coefficients. In linear regression, if there are two correlated features, there coefficients can be poorly determined and have high variance. One of them can have a very large positive coefficient, and the other correlated feature can have a very large negative coefficient. They cancel each other. By adding the ridge penalty, the problem is alleviated, as it shrinks the coefficients towards 0. In the extreme case of k identical features, they each get small identical coefficients. So ridge penalty encourages features to borrow strength from each other. From a Bayesian point of view, the ridge regression estimation assumes that w_j has a Gaussian distribution with 0 mean as its prior distribution. And the solution to ridge regression is the mean (or mode) of the posterior distribution.

Lasso behaves differently than Ridge. If there are several correlated features, Lasso tends to pick one and ignore the rest. That is, some features will have coefficients exactly 0. So Lasso can be used to perform continuous feature selection. From a Bayesian point of view, the Lasso penalty corresponds to a Laplace prior.

To illustrate the behaviors of Ridge and Lasso, we write them as constrained optimization problems.

Ridge regression can be equivalently formulated as

$$\hat{w}^{ridge} = \arg\min_{w} \sum_{i=1}^{N} (y_i - w^0 - \sum_{j=1}^{d} x_i^j w^j)^2$$

$$\text{subject to } \sum_{j=1}^{d} w^{j2} \leq t$$

There is a one-to-one correspondence between λ and t.

Lasso regression can be equivalently formulated as

$$\hat{w}^{lasso} = \arg\min_{w} \sum_{i=1}^{N} (y_i - w^0 - \sum_{j=1}^{d} x_i^j w^j)^2$$

$$\text{subject to } \sum_{j=1}^{d} |w^j| \leq t$$

Similarly, there is a one-to-one correspondence between λ and t.

Figure 1 shows the difference between lasso and ridge regression estimations when there are only two features. The square loss has elliptical contours. Ridge regression has a disk constraint region, while lasso has a diamond constraint region. In both constrained optimization problems, the optimal solution is the

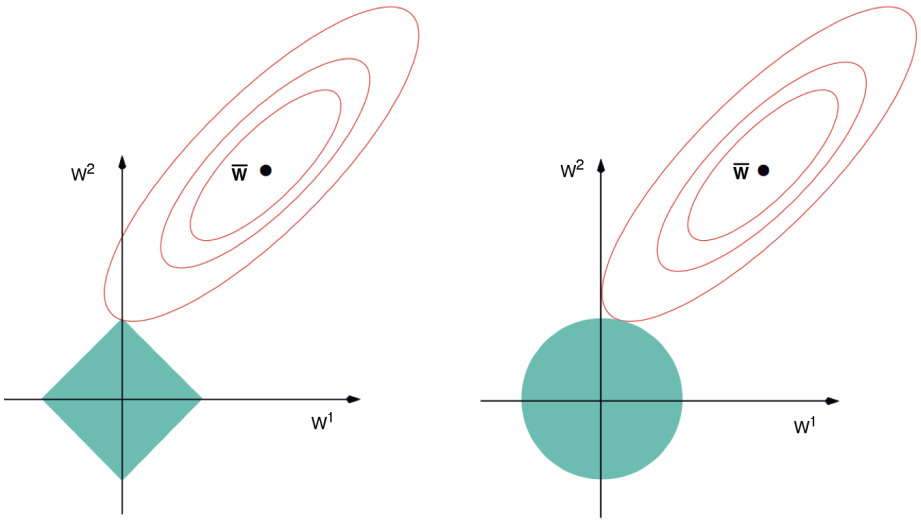

Figure 1: Source: Figure 3.11 of [4] Estimation picture for LASSO (left) and RIDGE (right). Solid areas are for regions of constraints $|w^1|+|w^2| \leq t$ (LASSO, left) and $(w^1)^2+(w^2)^2 \leq t$ (RIDGE, right). Red ellipses are the contours of the objective, here the least square function.

first point where the elliptical contours hit the constraint region. In Lasso regression, if the solution occurs at a corner, then it has one parameter w_j equal to zero. When d is large, there are many corners so that many parameters are likely to become zero.

For Ridge, the constraint region is a multidimensional sphere, so the elliptical contours can hit anywhere for a solution. We wont likely have zero w parameters, but likely none of them is large.

3.2 RIDGE-Regularized Logistic Regression

The regularized Logistic Regression has a maximum likelihood objective, thus the form

$$\max_{w_0,w} \frac{1}{N} \sum_{i=1}^{N} \left[y_i \log(P(y=1|x_i) + (1-y_i) \log(P(y=0|x_i) \right] - \frac{\lambda}{N} R(w)$$

If we use RIDGE penalty for $R(w)$, and the logistic predictor $h_w(x)$ as probability $P(y=1|x_i)$, we get $J(w)$ as following:

$$\max_{w_0,w} \frac{1}{N} \sum_{i=1}^{N} \left[y_i \log h_w(x) + (1-y_i) \log(1-h_w(x)) \right] - \frac{\lambda}{2N} \sum_{j=1}^{d} w^{j2}$$

Differentiating $J(w)$ by w components we get a gradient that is as before (see Logistic Regression), except now there is an extra term due to the RIDGE

penalty:

$$\frac{\delta J}{\delta w^j} = \frac{1}{N}\sum_{i=1}^{N}(y_i - h_w(x_i))x_i^j + \frac{\lambda}{N}w^j \texttt{ for any j = 1:d}$$

This gives the Gradient Descent update rule for regularized Logistic Regression:

$$w^0 := w^0 - \alpha\frac{1}{N}\sum_{i=1}^{N}(h_w(x_i) - y_i)x_i^0$$

$$w^j := w^j - \alpha\big[\frac{1}{N}\sum_{i=1}^{N}(h_w(x_i) - y_i)x_i^j + \frac{\lambda}{N}w^j\big] \texttt{ for any j = 1:d}$$

where α is the learning rate.

3.3 Solving Regularized Linear Models

packages:

- Liblinear[1]
- glmnet[2]
- sklearn[5]

References

[1] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

[2] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[3] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[4] Trevor Hastie, Robert Tibshirani, Jerome Friedman, T Hastie, J Friedman, and R Tibshirani. *The elements of statistical learning*, volume 2. Springer, 2009.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.