Predicting the Number of Shares of a Webpage

Alexandre Piche
260478404

Philippe Nguyen
260482336

Yash Lakhani
260500612

***Abstract*—We implementated a linear regression using the closed form, and the gradient descent solutions. We also used different shrinkage and dimension reduction algorithm to avoid overfit.**

I. Introduction

In the advertising industry it is of interest of predicting the popularity of a website to adequately set the price of ads on a given web page.

Simple tools like OLS have a surprising power, particularly when couple with regularization techniques such as the lasso or ridge.

II. Implementation of OLS

$$Y = X\beta + \epsilon \tag{1}$$

A. Closed Form

With the traditional assumption of $X^T\epsilon = 0$ [1], i.e. that the error is uncorrelated with the matrix X, it is easy to solve for the weights, the resulting equation is given by

$$\begin{aligned} Y &= X\beta + \epsilon & (2)\\ X^TY &= X^TX\beta + X^T\epsilon & (3)\\ \hat{\beta} &= (X^TX)^{-1}X^TY & (4)\end{aligned}$$

B. Gradient Descent

It is computationally inefficient to invert large matrices such as the one provided for this exercise. It is more efficient to minimize the sum of squares $SSR(\beta) = \sum_{i=1}^{n}(y_i - \mathbf{x}_i\beta)^2$. We need to take the derivative to

$$\frac{\partial SSR(\beta)}{\partial \beta} = -2X^T(Y - X\beta) \tag{5}$$

cite Joelle's slides lecture 2

while $\epsilon > 0.1$ and $i <$ max_iterations **do**
  hypothesis $\leftarrow X^T\beta$
  loss $\leftarrow$ hypothesis - Y
  gradient $\leftarrow$ 2 X^T loss
  $\beta_{\text{new}} \leftarrow \beta - \frac{\alpha * \text{gradient}}{n}$
  $\epsilon \leftarrow \|\beta_{\text{new}} - \beta\|$
  $i \leftarrow i + 1$
  $\beta_{\text{new}} \leftarrow \beta$
end while

C. Supplementary information about the Weights

Only having the size of the weights is of little used if we cannot asses the fit of the model.

1) Signifiance of our Weights: It is possible that even if the weight is big that the variance is so high that it is not significantly different from zero.

2) Adjusted R^2: To asses the fit of the model, it is helpful to look at the percentage of the sum of square that is explained by our model.

3) AIC and BIC: Adding a variable cannot decrease the fit and almost always increase the prediction power of the model. However it might add noise to the prediction. AIC and BIC are a way to quantify the importance of adding a feature to our model by penalizing complexity.

III. Model Complexity and Dimension Reduction

There is a total of ?? features, which seems like a lot. Given the high number of features, it might be preferable to reduce the dimension to achieve better out of sample prediction.

The widely accepted Occam's razor principle also suggests that parsimonous models generalize better than more complex model.

There is multiple way to avoid an over complex model that will generalize well for prediction, we will explore the performance of some of the most known of them.

A. Principal Component Analysis

Given the large dimension of the dataset and that some of the feature are highly correlated we decided to the principal component analysis algorithm to reduce the dimension by using principal component analysis (PCA) algorithm. We noticed that over 95% of the variance can be explained by the first 3 dimensions. The idea behind the PCA algorithm is trying to reconstruct X, by the minimal set of component. Namely we want to find a W such that

L linear basis vector

X is $K \times N$, where K is the number of feature and N the number of examples.

$$J(W, Z) = \|X - WZ^T\|_F^2 \tag{6}$$

Where W is $K \times L$ orthonormal and Z is $N \times L$ matrix [2] [3]

B. Ridge or L2-Regularization

$$\hat{\beta}^{\text{ridge}} = \underset{\beta}{\text{argmin}} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\} \quad (7)$$

[4]

The gradient will then be

$$\frac{\partial SSR(\hat{\beta}^{\text{ridge}})}{\partial \hat{\beta}^{\text{ridge}}} = -2X^T(Y - X\beta) + 2\lambda\|\beta\| \quad (8)$$

We can then add the following condition to our gradient descent algorithm

if 'Ridge' is True **then**

loss $+= 2 * \lambda\|\beta\|$

end if

Ridge regression can also be incorporated fairly easily to OLS

$$w = (X^T X + \lambda I)^{-1} X^T Y \quad (9)$$

Here, λ is a hyper-parameter that we can change and optimize using cross validation. A larger λ gives a greater amount of regularization. Having $\lambda = 0$ means no L2 regularization, giving the standard ordinary least squares solution.

C. Lasso or L1-Regularization

Lasso can be used for features selection by setting some of the coefficients to zero. Note that we should be careful in dropping features because it might increase the bias of our model, however it will reduce it's variance.

$$\hat{\beta}^{\text{lasso}} = \underset{\beta}{\text{argmin}} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\} \quad (10)$$

There's no closed form solution for the lasso, since the solution is non-linear (due to the absolute value) [4]. We used the python library "Scikit Learn" [3] to estimate the lasso parameters.

D. Feature Normalization

It was numerically challenging to apply the gradient to feature that are of multiple order different from each other. We can normalize without changing their span, since it is a linear transformation.

IV. Cross-Validation

complete randomization of the fold, by a random variable

Our cross validation is done by choosing a k, which is the number of cross validation folds. We split the entire data set into k different splits, such that each split has the same number of instances. If the data set is not divided equally by k, the remainder is added to the last split. We shall use $X[i]$ and $Y[i]$ to denote split i of the data set.

for fold i **do**

Set $X[i]$ to be the validation feature matrix

Set $Y[i]$ to be the validation response vector

Combine all other $X[-i]$ to be the training feature matrix

Combine all other $X[-i]$ to be the training response vector

for every hyper-parameter setting **do**

Train the linear model with the current training set and given hyper-parameters

Generate a prediction for Y given the validation feature matrix $X[i]$

Record the error between the predicted $\hat{Y}$ and $Y[i]$

end for

end for

Average the error for each hyper-parameter setting across every fold

Choose the hyper-parameter setting with the lowest average error

A. Hyperparameters Optimization

Feature selection using the lasso function from [3]

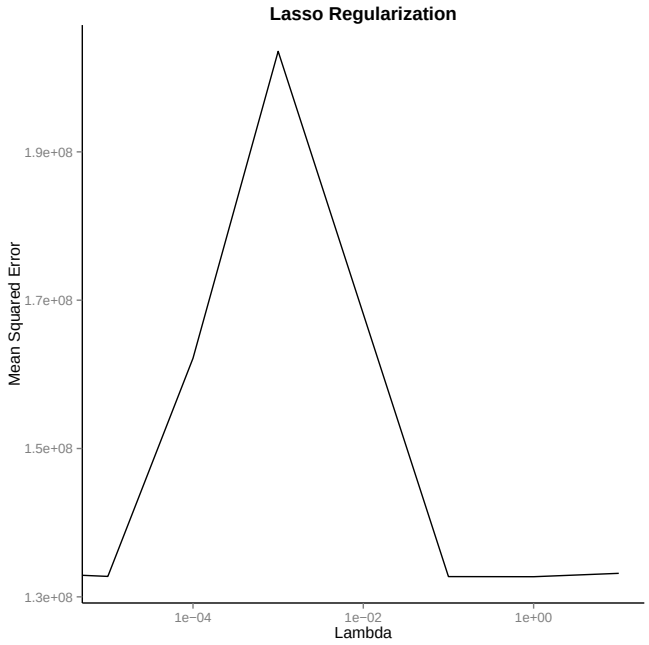
Trying to avoid overfitting to be able to generalize to new examples.

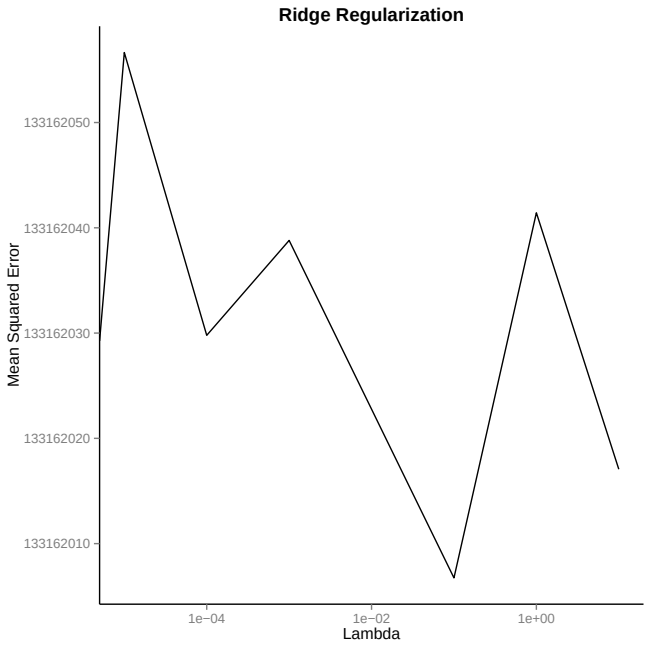
we want to optimize the learning rate and the penalty rate

V. Results

Also talk about the α parameter for the gradient descent.

Talk about the mean squared error (MSE) obtain when we varied the alpha of the lasso





Algorithm	Mean Squared Error
OLS	132736810
Lasso	132712496
Ridge	133162007
Principal component analysis	10C

Given the small difference it might be worth to take the difference in results with a pinch of salt, since it might not be significant.

VI. Complementary Datasets

Huffington post

VII. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

References

[1] R. Davidson and J. G. MacKinnon, *Econometric theory and methods*. Oxford University Press New York, 2004, vol. 5.

[2] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[3] 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*, vol. 12, pp. 2825–2830, 2011.

[4] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, "The elements of statistical learning: data mining, inference and prediction," *The Mathematical Intelligencer*, vol. 27, no. 2, pp. 83–85, 2005.