MO444 1s2018 - First Assignment

Lucas Rath Maia, RA:117751*

1. Introduction

In this report, corresponding to the first assignment of the course "Machine Learning and Pattern Recognition", linear regression alternatives will be explored in order to predict the number of shares in social networks (popularity) of a certain data set. Starting with a simple linear regression, this report will also come up with more complex alternatives in order to reduce the prediction error.

Furthermore, the data set used consists of information from 39,000 articles published by the Mashable website. This information was then processed by Fernandes [1] and 60 relevant features were extracted.

2. Linear Regression

2.1. Normal Equation

Linear regression is a mathematical model that, concerning its weights $\boldsymbol{\theta} \in \mathbb{R}^{1\times n}$, linearly correlates the inputs to the outputs and has the general form of

$$h_\theta(\boldsymbol{x}^{(i)}) = \boldsymbol{x}^{(i)} \cdot \boldsymbol{\theta} = \theta_0 + x_1^{(i)}\theta_1 + ... + x_n^{(i)}\theta_n \quad (1)$$

where h_θ is the function output and $x^{(i)}$ a input vector of $n+1$ elements when considering $x_0^{(i)} = 1$. Additionally, when evaluating the model output over all m samples, we can define the predicted output vector as:

$$H_\theta(X) = X \cdot \boldsymbol{\theta} = [h_\theta(x^{(1)}), ..., h_\theta(x^{(m)})]^T \quad (2)$$

where $X \in \mathbb{R}^{m\times(n+1)}$ is the input matrix of n features and m samples. We then define a a cost function $J(\theta)$ over all the m samples, which we aim to minimize:

$$\min_\theta J(\boldsymbol{\theta}) = \min_\theta \frac{1}{2m} \left\| Y - H_\theta(X) \right\|^2 \quad (3)$$

where $Y = [y^{(1)}, ..., y^{(m)}]^T$ is the observed output vector. Furthermore, it is known that if h_θ is a linear function regarding θ the cost function is then convex and therefore there is only one minimum, which is also the global minimum. This point can be also obtained by a closed formula, which for sake of simplicity, it will be shown directly:

$$\boldsymbol{\theta} = (X_{tr}^T X_{tr})^{-1} X_{tr}^T \, Y_{tr} \quad (4)$$

Where the subindex *tr* means that the samples are consisted only of training data. Further, a more complex linear regression model is the quadratic form:

$$h_\theta(\boldsymbol{x}^{(i)}) = \theta_0^b + x^{(i)T} \begin{bmatrix} \theta_1^c \\ \vdots \\ \theta_n^c \end{bmatrix} + x^{(i)T} \begin{bmatrix} \theta_{1,1}^q & \cdots & \theta_{1,n}^q \\ 0 & \ddots & \vdots \\ 0 & 0 & \theta_{n,n}^q \end{bmatrix} x^{(i)} \quad (5)$$

While there are nonlinear combination of the input features, we still have a linear function concerning all the adjustable parameters θ, and therefore the linear regression is still possible but now with a higher complexity to adjust the data. Therefore, this new function can be also viewed as $h_\theta(x^{(i)}) = \hat{x}^{(i)} \cdot \boldsymbol{\theta}$, where $\hat{x}^{(i)}$ is the old input vector $x^{(i)}$, but now mapped to a higher feature space.

2.2. Gradient Descent

A second method of calculating the weight vector is called *gradient descent*, which is an iterative method based on stepping toward the opposite gradient direction. This method has the advantage of not requiring to calculate matrices inversion, however it can get easily stuck in a local minimum. The algorithm starts with a random initialization of the weight vector and at each step the weights are updated according to the learning rate α and the gradient calculated by deriving the cost function J in equation 3:

$$\boldsymbol{\theta}_j = \boldsymbol{\theta}_j - \alpha \frac{\partial J}{\partial \boldsymbol{\theta}_j} = \boldsymbol{\theta}_j - \frac{\alpha}{m} \left(Y_{tr} - X_{tr}\boldsymbol{\theta}\right) \boldsymbol{x}_{j,tr} \quad (6)$$

In fact, the learning rate an very important parameter and its value can change the algorithm performance drastically. When choosing a small value, the cost function can take a long time until it converges to the minimum. On the other hand, big values can cause the effect of *overshoot*, when the cost function swings around the minimum and taking a long time to converge, or even worse, can diverge. To deal with this problem, Zeiler [2] proposed one of the most famous methods ADAGRAD to dynamically determine the learning rate for each weight depending on all the previous gradients calculated:

$$\alpha_j^t = \frac{\alpha_{global}}{\sqrt{\sum_1^t (g_j^t)^2}} \quad (7)$$

where g_j^t is the cost function gradient with respect to the weight j at the *t-th* iteration. In this way, the dynamic learning rate will grow with the inverse of the gradient magnitudes. This will ensure that it makes a large step when the cost function appears to plateau and will make small steps toward large gradients. As desired, the rate will also decrease over the course of iterations.

*Is with the Faculty of Mechanical Engineering, University of Campinas (Unicamp). **Contact**: lucasrm25@gmail.com

3. Dealing with Overfitting

As the model become excessively complex, there is always a risk of overfitting the data. Therefore, a good regression model is a model which presents the capability to generalize from trend rather than memorizing data. In order to deal with this problem, we first need to be able to identify when a model is overfitting or not. The more intuitive solution is to divide the available data into 3 different subsets: 60% training, 20% validation and 20% test data, and use the prediction quality related to each subset as overfit indicators. A simple illustration of overfitting during the training can be seen in the figure 1.

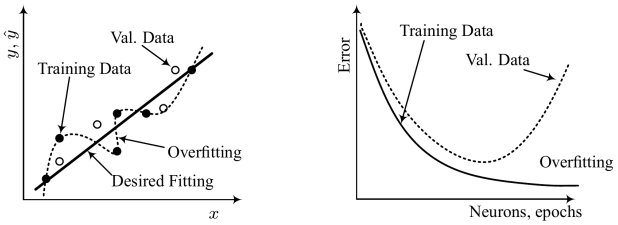

Figure 1. Overfitting of a Metamodel

asd'asd

It is now easy to see that, as the model increases its complexity or becomes more flexible, the prediction quality regarding the training data will decrease even more. At the beginning, the validation data quality will also decrease, as the model learns better the relation between the input and output. This first phase is called underfitting. However, at some point, the prediction quality related to the validation data will starts to increase and then we have the so called overfitting. The model is then loosing the generalization ability. In this context, the same logic could be applied if, in the figure 1, the x-axis were labeled as "model complexity" and the validation-data replaced by test-data.

As a conclusion, the desired model, which does not underfit or overfit the data, presents the lowest possible validation error during training and the lowest possible test error concerning different model complexities (number of model adjustable parameters).

In order to measure the prediction error of some data subset, besides the already presented *Mean Square Error* by J in equation 3, the *Coefficient of Determination R2* will be also used:

$$R^2 = 1 - \frac{\|Y - H_\theta(X)\|^2}{\left\|Y - \overline{Y}\right\|^2} \tag{8}$$

Where $\overline{Y}$ is the mean of the observed input. As noted, the *MSE* $\in [0, \infty)$, so that 0 means that the model has perfectly predicted the observations and it goes to infinity as the predicted output diverges from the observed data. On the other hand, *R2* $\in (-\infty, 1]$ and it goes to 1 when the prediction is perfect. In particular, we use *MSE* for reporting because it is a simple metric and it is technically the loss-function, but *R*2 is useful because it is often easier to interpret since it is normalized and does not depend on the scale of the data.

3.1. Regularization

In this context, a technique called *regularization* fights against overfitting by penalizing the weight vector norm:

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\left(\|Y - H_\theta(X)\|^2 + \lambda \sum_{i=2}^{n+1} \theta_i^2\right) \tag{9}$$

and the closed formula can be also derived resulting in:

$$\boldsymbol{\theta} = \left(X_{tr}^T X_{tr} + \lambda \begin{bmatrix} 0 & 0 \\ 0 & I^{n\times n} \end{bmatrix}\right)^{-1} X_{tr}^T\, Y_{tr} \tag{10}$$

where λ is the regularization factor. Big λ will produce a small weight vector norm and will fight with overfitting, resulting in a biased model. On the other hand small λ values will encourage a model with high variance. But how to find the right value for λ? As already explained, the right value is the one, which produces the lowest prediction error related to the validation data. Accordingly, independently from the chosen feature space, he have a second minimization problem:

$$\lambda = \underset{\lambda}{\operatorname{argmin}} \frac{1}{2m} \|Y_{va} - X_{va}\boldsymbol{\theta}\|^2 \tag{11}$$

Next, it is known that θ can be obtained directly using the training data through normal equations, according to formula 4. Finally, a direct equation can be derived:

$$\lambda = \underset{\lambda}{\operatorname{argmin}} \frac{1}{2m} \left\|Y_{va} - X_{va} \cdot (X_{tr}^T X_{tr} + \lambda I)^{-1} X_{tr}^T\, Y_{tr}\right\|^2 \tag{12}$$

which can be obtained through a unidimensional search method, such as derivative approximation methods. However, if desired the regularization coefficient can be adjusted by hand taking into account the balance between bias vs variance.

4. Enhancing Model Quality

4.1. Data Normalization

When the scale of some features dominate the others, the GD method can take longer to converge. In this way, data normalization is important since it scales the features to the same range of values and therefore the gradients will also have similar magnitudes. Among many normalization techniques, the mean normalization will be used:

$$\hat{\boldsymbol{x}}_j = \frac{\boldsymbol{x}_j - \mu_j}{\sigma_j} \tag{13}$$

where the feature j will be scaled by its mean μ and standard deviation σ. The model is then trained with normalized inputs and outputs, but remapped again to the original space

4.2. Feature Selection

Another popular technique commonly used is the feature selection, which by eliminating a subset of irrelevant features, enhances the model quality and speeds up the training process. A very popular method is called *backward elimination* [3], so that features are eliminated one by one. The removed feature is the one that, when removed, the remaining subset have the smallest mean square error (MSE).

4.3. Dealing with Discrete Variables

In the presented data set are, as usual, discrete variables that must be properly treated in order to ensure a better prediction quality. As a rule, the discrete data must be orthogonal to each other, i.e. the distance measured two by two must be always the same. In this way, when representing the input referent to the week day of the publication, this information could be represented as a single discrete input belonging to the subset $1, 2, 3, 4, 5, 6, 7$. However, this representation would not be orthogonal. Thus, a more suitable way is representing the data as 7 separate binary inputs. Obviously, this is only adequate, since a publication is not published in more than one week day at the same time.

5. Results and Discussion

Starting with the simplest solution, a linear and quadratic linear regression was performed through normal equations and the results can be seen in table 1. As noticed, the model prediction quality obtained is very bad for both linear and quadratic model and for both training and testing data. Although the error related to the quadratic model has decreased a little, the error is still very big. At a first sight, this means that a simple linear regression is not flexible enough to represent the data, which seems to present a highly complex relation between input and output.

Table 1. NE prediction performance of a simple LR model

	MSE_{tr}	MSE_{te}	R^2_{tr}	R^2_{te}
linear	1.122 e8	2.124 e8	0.03	0.01
quadratic	1.098 e8	2.151 e8	0.05	0.00

Although we now that our model is not overfitting since both training and testing error are too high, a regularized model was created using the optimized closed equation 12. A λ of 0.96 was obtained and the results are shown in table 2.

Table 2. NE prediction performance of regularized LR model

	MSE_{tr}	MSE_{te}	R^2_{tr}	R^2_{te}
quadratic-regul.	1.098 e8	2.151 e8	0.05	0.00

As expected, the model quality has almost not changed. This is due to the very low λ found and to the fact that the model is highly biased and underfitted.

Now, let's explore some of the gradient descent procedures. As explained, the defined linear regression cost function is convex and therefore has only one local minima. Performing a *GD* using the quadratic form and a initial learning ratio of 1e-25, the following MSE over iterations can be seen in figure 2.

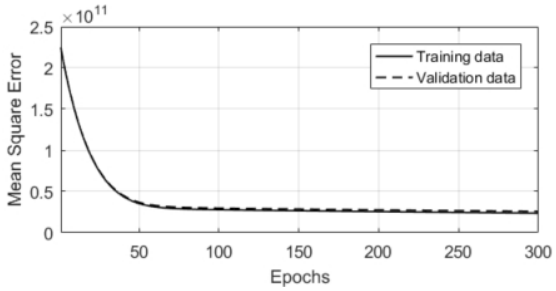

Figure 2. GD for the quadratic linear regression

Initially, it was almost impossible to find a proper α value, so the cost function did not diverge and at the same time did not converge too slow. For this reason, a new strategy was implemented: if the norm of the gradients concerning all weight has increased between one iteration and another, the actual learning rate was divided by 10. Even so, the cost function used to diverge.

As a next improvement, the data was first normalized and the *GD* method was performed again. The chosen strategy avoided the cost function to diverge for most initial learning rates chosen, the error has converged faster and to a lower value.

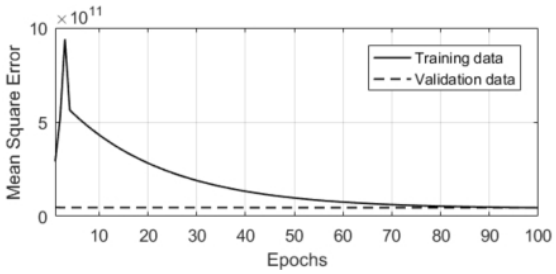

Figure 3. GD for the quadratic linear regression - normalized

However, the cost function is still far away from the normal equation result. One reason is because the NE method uses both training and validation data to train the model while the GD uses only the training data keeping the validation data as a overfitting indicator. In this way, more data might result in a slight prediction improvement. Additionally, the validation error lower than the training error is an indicator that our model quality is so poor and underfitted that it is sometimes predicting better the unknown validation data than the known data.

On the issue of convergence, even better is the ADAGRAD method of adjusting dynamically the learning rate. Notably, the cost function converged for any initially chosen α and has converged very fast to the minima.

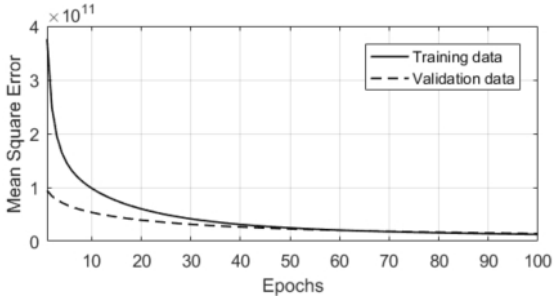

Figure 4. GD for the quadratic linear regression - ADAGRAD

Further, as can be seen in figure 5, the ADAGRAD method is also very robust. Even when choosing a very big learning rate, the method kept the control and avoided the error to explode and diverge, what was not so easy and intuitive with the previous methods.

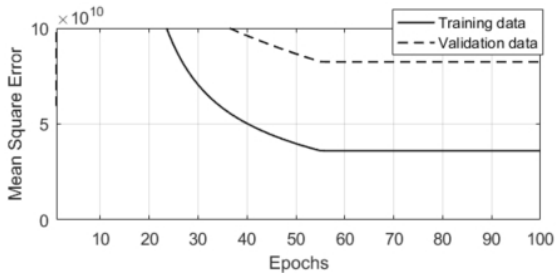

Figure 5. GD for the quadratic linear regression - ADAGRAD with higher initial learning rate

Another important point to be noticed is that the model has a very bad prediction quality and anyway, the model is not overfitting, since the validation error is always decreasing. Moreover, the prediction quality is so bad, that the validation error is sometimes smaller than the training error.

Finally, a last improvement in the model was done. It was already explained that features with low correlation with the output can slow down the training process and can lead in the end to worse prediction quality. Performing then the *backward elimination method*, the following features were identified as the 15 less relevant.

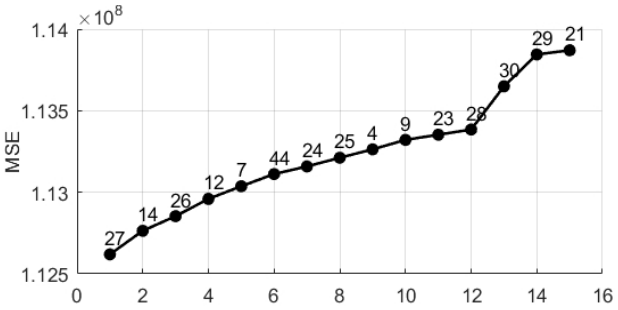

Figure 6. Backward elimination for quadratic linear regression using GD

The figure 6 shows that no improvement in the model prediction has been done when removing features since the error has always increased as the features had been eliminated. In theory, when dealing with normal equations, as we always found the global minima, removing features will not decrease the prediction error if we have enough data. However, an improvement in the gradient descent convergence speed can still be obtained since the search space will be smaller, making it easy to converge and find the minima. After performing the feature selection, the results are briefly shown in table 3.

Table 3. Prediction performance of a LR model with feature selection using NE

	MSE_{tr}	MSE_{te}	R^2_{tr}	R^2_{te}
quadratic	1.193 e8	2.167 e8	0.03	-0.01

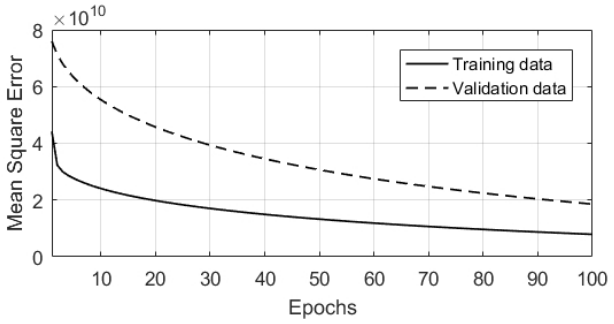

Figure 7. GD iterations for quadratic linear regression with feature selection

As expected, the prediction performance using normal equations became worse. However, as seen in figure 7, the error for the gradient descent method has decreased over iterations, when comparing with the model with all the 58 features.

Anyway, it is very difficult to make concrete assumptions about the improvement because the error is still absurdly high and the model is highly underfitted. Moreover, when training the model different times, complete different error curves are observed. Often, the error related to the validation data is lower than the related to the training data, what again, absurdly indicates that the model is predicting better the unknown validation data than the learned training data.

6. Conclusions

Linear regression alternatives have been implemented and tested in order to predict the number of shares of publication from the Mashable website. Solutions involving data normalization, regularization and feature selection were presented using both normal equations and gradient descent algorithm.

Unfortunately, the linear regression and its more complex variations were not able to predict the output with accuracy. A simple linear regression did not appear to be flexible enough to represent the data, which seems to present a highly complex relation between input and output.

For a possible better prediction, non-linear models, such as variations of the multi-layer-perceptron trained using backpropagation techniques would probably reduce drastically the error.

References

[1] P. Vinagre K. Fernandes and P. Cortez. A proactive intelligent decision support system for predicting the popularity of online news. In *Proceedings of the 17th EPIA 2015 - Portuguese Conference on Artificial Intelligence*, September 2015. 1

[2] Matthew D. Zeiler. Adadelta: An adaptive learning rate method. In *New York University, USA; Google Inc., USA*, 2012. 1

[3] I. Guyon. An introduction to variable and feature selection. In *Journal of Machine Learning Research 3*, pages 1157–1182. 2