MC886 - Assignment 1
Predicting social network shares using Linear Regression

Gustavo de Mello Crivelli*
RA 136008

Abstract

Using a dataset of over 30.000 entries [1], Python was used to build and train a predictor for the number of times a given publication will be shared in social networks. A separate dataset was then used for testing. An initial attempt with only basic linear regression was able to make predictions with an error of 3183 shares, and managed to predict 2.6% of the test dataset with an error of 100 shares. Further optimizations using Gradient Descent, feature scaling and regularization led to a final predictor able to guess shares with an error of 2788 shares, and had 8.3% of predictions within 100 shares of the correct value.

1. Introduction

Social networks are bustling hubs of information exchange. Being able to predict how far an article might go in terms of social network sharing, as well as understanding what makes a publication become viral, is invaluable for any content creator.

This report details the process of attempting to build a predictor able to do the above, using a range of Linear Regression and Gradient Descent techniques.

1.1. Dataset

The dataset used for this work was published by www.mashable.com [1] [2]. Their content as the rights to reproduce it belongs to them. Hence, this dataset does not share the original content but some statistics associated with it.

The training dataset contains 31715 entries, each with 58 total predictive attributes and a target attribute (the number of shares). Each feature has a different value range, and some of them are discrete. All of this needs to be dealt with in order to achieve better results in prediction.

The test dataset contains 7929 entries and is similar to the training dataset in terms of attributes. It is used to validate the predictor learned while using the training dataset.

2. Proposed Solutions

All results are listed in the Results and Discussion section.

2.1. Linear Regression

The first attempt was made using basic linear regression. Given a dataset X with m entries, an entry $x^{(i)}$ from that dataset and a target value $y^{(i)}$, we want to find an array of coefficients θ such that, given $h_\theta(x) = \theta \cdot x$, the cost function $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$ is minimal [3].

For this, scikit-learn [4] was used. No further optimizations were done at this time.

2.2. Feature Scaling

The second proposed solution utilized the same linear regression as above, but normalized the features in the training to keep their values in the range $[0, 1]$. The test dataset was also normalized, using the same parameters found during the training set normalization.

Surprisingly, this performed with worse results than the non-normalized attempt.

2.3. Gradient Descent

The third proposed solution moved on to Gradient Descent. Let n be the number of coefficients in the θ array. In order to find the minimum cost of $J(\theta)$, the coefficient array θ is iteratively updated until the function converges. This is described by $\theta_j := \theta_j - \alpha\frac{\delta}{\delta\theta_j}J(\theta)$ for $0 \leq j < n$, where α is a scalar representing the learning rate.

The feature scaling proposed above was also used in this and in the following proposed solutions. For this solution, 10000 iterations were made, and an initial θ was built with random values in the range $(-1000, 1000)$.

Figure 1 details the variations in the mean absolute error of predicted shares as the training iterates. The high values are a good indication of overfitting to the training dataset, and this will need to be dealt with.

*Undergrad student in the Institute of Computing, University of Campinas (Unicamp). **Contact**: gmcrivelli@gmail.com

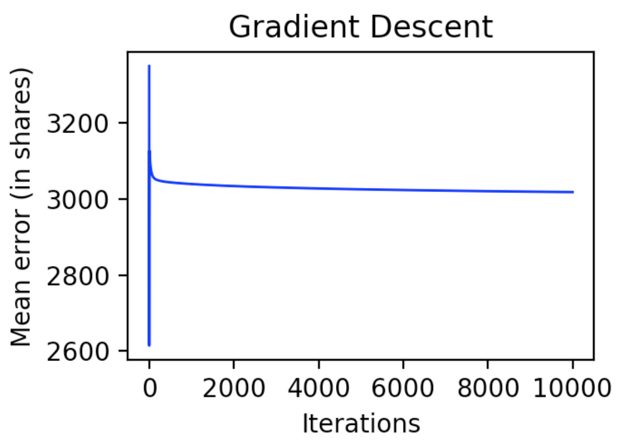

Figure 1. Mean error by Gradient Descent iteration.

2.4. Gradient Descent with Feature Dropping

The next step was to remove superfluous features that might be contributing with noise, and leading the predictor to overfit to the training data. For this, the coefficient array found in the previous round of training was analyzed, and every feature whose corresponding coefficient had an absolute value inferior to 1000 was removed from consideration.

After this procedure, the only features still in consideration were the ones in the set $F = 6, 8, 11, 18, 22, 23, 24, 25, 26, 27, 28, 29, 30, 29, 41, 43$. (Please reference the dataset [1] for more details.

Furthermore, the dataset itself was filtered, and entries with anomalous features were removed. For instance, all entries marked as having zero words in the content (Feature 3), were likely the result of a malformed observation, as can be easily verified by accessing the link to the publication in that entry.

The results of these changes can be seen in Figure 2

2.5. Gradient Descent with Regularization

It was then attempted to build a more complex predictor by changing $h_\theta(x)$ to be a second-degree polynomial function. For this, a new feature array was produced for each entry, composed of an array of the features mentioned in the previous step, concatenated with an array containing the square of those features.

To minimize overfitting, a λ regularization parameter was introduced, so that the cost function becomes $J(\theta) = \frac{1}{2m}(\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2)$. The intent of this is to penalize more complex models over simpler ones, in a way that will reinforce important features while reducing the influence of noise.

The lambda with the best results was found to be $\lambda = 1000$. The results of these changes can be seen in Figure 3

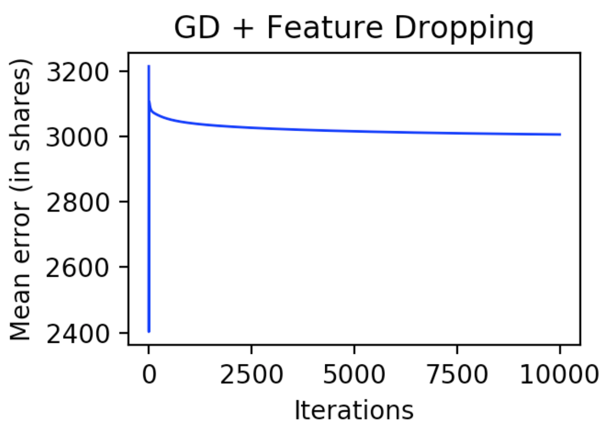

Figure 2. Mean error by Gradient Descent iteration, with feature dropping.

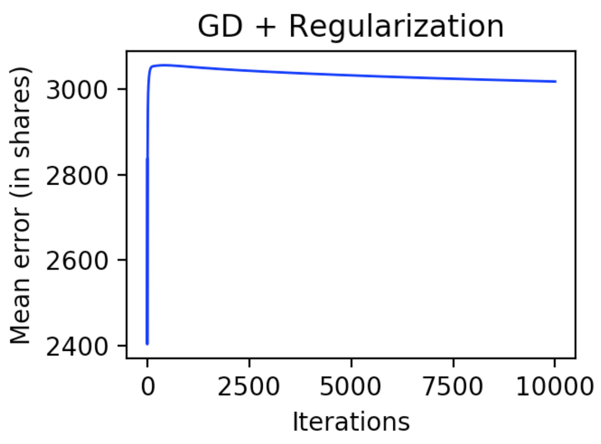

Figure 3. Mean error by Gradient Descent iteration, with feature dropping and regularization.

2.6. Gradient Descent with Further Feature Dropping

A second attempt was made at choosing the most relevant features. Instead of only taking the features with the highest coefficients, the following algorithm was executed:

1. Start with an empty set of selected features F.
2. For every feature f in the dataset that is still not in F, train and evaluate the predictor using only $F \cup \{f, f^2\}$ as features.
3. Select the feature f that resulted in the lowest mean error in Step 2. If it is lower than the previous mean error, add $\{f, f^2\}$ permanently to F. Otherwise, stop.
4. Repeat from Step 2.

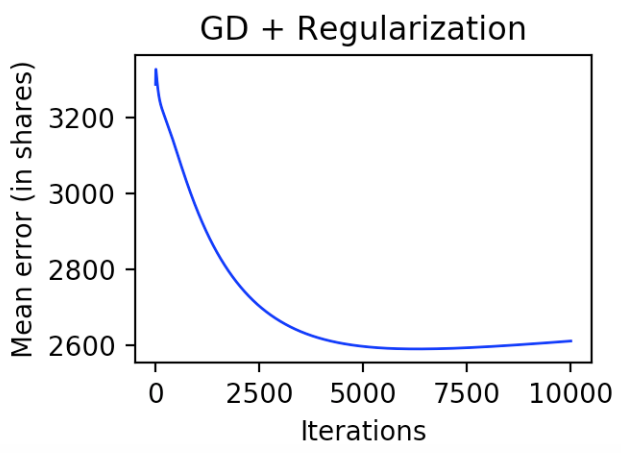

Figure 4. Mean error by Gradient Descent iteration, with further feature dropping and regularization.

Using this, a new set of features was selected, composed of features $F = 6, 7, 8, 19, 21, 25, 27, 29, 35$. This takes care of the discrete features, as the only remaining discrete feature is Feature 25 (whether or not the article was published on a Friday). The results of this filtering can be seen in Figure 4, and are considerably better than in the previous attempts.

2.7. Normal Equations

At this point, a different direction was attempted by using Normal Equations to analytically solve the regression. All of the techniques adopted regarding dataset filtering and feature processing were also applied here.

Since calculating θ in this manner requires a lot of computing power, a smaller sample of the original dataset was used. This contained the first 10000 entries that were considered to be usable.

3. Results and Discussion

Table 1 contains the results of the above experimentation.

Surprisingly, every technique implemented seemed to deliver progressively worse results. This may have been because of the amount of noise contained in the dataset, as is suggested by the drastic improvement in the results once the features were re-analyzed and re-selected.

The results obtained when Normal Equations were attempted indicate that it is able to reach the same results as the Gradient Descent method. However, due to its increased demand of memory and computing power, such a method is only feasible by either selecting a smaller subset of training examples, or by some other manner of training in batches that may allow the usage of all training examples (although this was not done here).

As for the learning rates applied in Gradient Descent implementations, it was found through experimentation that for implementations 2.3, 2.4 and 2.5, the optimal learning rate was $\alpha = 0.1$, as larger values caused the training to diverge and smaller values were simply slower to converge. For implementation 2.6, the optimal learning rate was found to be $\alpha = 0.05$.

4. Conclusions and Future Work

Linear regression and Gradient Descent are the most basic of Machine Learning tools. However, it is possible to increment those with more advanced techniques in order to achieve better results, as was shown by the results obtained by experimentation.

That it was possible to predict shares with a mean absolute error of 2789.5 leads to a few conclusions. First, predicting social network shares is absolutely possible, as the predictor did get better at it as more techniques were implemented.

Second, it may be that the current model is not complex enough to properly predict shares, and more advanced techniques such as Deep Learning are likely to deliver better results.

Finally, if you are going to publish something, it is probably a good idea to do it on a Friday.

References

[1] www.mashable.com. Dataset of popularity in social networks. *https://www.dropbox.com/s/hglzupqi2aubs36/data.zip?dl=0*. 1, 2

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

[3] Prof. Anderson Rocha. Classroom material for mc886 - machine learning and pattern recognition. *https://www.ic.unicamp.br/ rocha/teaching/2018s1/mo444/index.html*. 1

[4] http://scikit learn.org/. Python library for machine learning. 1

	Mean error	**Accuracy within 500 shares**	**Accuracy within 100 shares**
(2.1) Linear Regression (LR)	3183.5	14.6%	2.6%
(2.2) LR + Feature Scaling	3306.2	13.8%	2.7%
(2.3) Gradient Descent (GD)	3194.3	11.5%	2.2%
(2.4) GD + Feature Dropping (FD)	3199.4	9.9%	2.0%
(2.5) GD + FD + Regularization (Reg)	3228.2	9.7%	1.9%
(2.6) GD + Reg + Further FD	2789.5	33.3%	8.3%
(2.7) Normal Equations	3176.5	17.7%	3.2%

Table 1. Comparison of the results obtained.