Article popularity prediction by linear regression. Can it be done?

Giuliano R Pinheiro
(RA 108579)

1. Introduction

The time when news websites and the likes were large collections of articles are long gone. After social media came along these content producers decided to make those vanilla articles richer by incorporating social tools.

Today you can read an article while you might select a portion of the text and tweet it, share it via any social network you can imagine (and another lot you didn't even think existed), highlight portions of the article and message the author privately or just open your mind in the comment section, as well as rate the article if you like it.

That's close to what happens at `mashable.com`, where readers have at their disposal several sharing options for he article they are reading and an interesting tool called "The Mashable Velocity Graph" where they can see how fast that article is being shared over time.

That called the attention of a group of researchers at Universidade do Porto[1]. They published the dataset I use in this study which consists of statistics describing thosands of Mashable articles an their number of shares.

The objective, like I put in the title, is to find out if a linear model, taught by gradient descent optimization, can predict the number of shares for a given article.

Let's find out, shall we?

2. State of the Art

The authors of the original study on this data[1] tackled the problem by an interesting angle. They tried to predict if an article would be popular or not according to some fixed threshold, therefore solving a classification problem.

On their study they also enlist the help of robust models like Random Forests, AdaBoost, SVM, KNN and Naïve Bayes alongside their almost 38000 articles database, that I use for my task.

Their pipeline involves a complete modularised solution including a heuristic that tries to find better solutions near the first they find by Local Search techniques like Hill Climbing and Monte-Carlo Random Search.

Their best results overall were with Random Forests, resulting in 67% accuracy with 71% recall, 69% F1 and 73% AUC.

3. Proposed Solution

In this study I try to predict the number of shares which, in contrast to the original study, is a regression task. The proposed solution tries to do this with a linear model driven by Gradient Descent as the optimizer.

The overall pipeline is:

- Rescale the data to avoid Gradient Descent sensibility to dominating features
- Select features to reduce dimensionality and, therefore, distances between articles
- Add complexity by the use of composite features up to a point (to be discussed)
- Train a linear regressor with Gradient Descent
- Predict shares on validation data

While GD optimized Mean Square Error (MSE), the scoring metric was the Mean Absolute Error (MAE), which is hereby referenced as error, loss or residual, unless specified otherwise.

Let's take a look at our data, starting by looking at what are the features and what they mean:

url:	URL of the article (non-predictive)
timedelta:	Days between the article publication and the dataset acquisition (non-predictive)
n_tokens_title:	Number of words in the title
n_tokens_content:	Number of words in the content
n_unique_tokens:	Rate of unique words in the content
n_non_stop_words:	Rate of non-stop words in the content
n_non_stop_unique_tokens:	Rate of unique non-stop words in the content
num_hrefs:	Number of links
num_self_hrefs:	Number of links to other articles published by Mashable
num_imgs:	Number of images
num_videos:	Number of videos
average_token_length:	Average length of the words in the content
num_keywords:	Number of keywords in the metadata

data_channel_is_lifestyle: Is data channel ’Lifestyle’?

data_channel_is_entertainment: Is data channel ’Entertainment’?

data_channel_is_bus: Is data channel ’Business’?

data_channel_is_socmed: Is data channel ’Social Media’?

data_channel_is_tech: Is data channel ’Tech’?

data_channel_is_world: Is data channel ’World’?

kw_min_min: Worst keyword (min. shares)

kw_max_min: Worst keyword (max. shares)

kw_avg_min: Worst keyword (avg. shares)

kw_min_max: Best keyword (min. shares)

kw_max_max: Best keyword (max. shares)

kw_avg_max: Best keyword (avg. shares)

kw_min_avg: Avg. keyword (min. shares)

kw_max_avg: Avg. keyword (max. shares)

kw_avg_avg: Avg. keyword (avg. shares)

self_reference_min_shares: Min. shares of referenced articles in Mashable

self_reference_max_shares: Max. shares of referenced articles in Mashable

self_reference_avg_sharess: Avg. shares of referenced articles in Mashable

weekday_is_monday: Was the article published on a Monday?

weekday_is_tuesday: Was the article published on a Tuesday?

weekday_is_wednesday: Was the article published on a Wednesday?

weekday_is_thursday: Was the article published on a Thursday?

weekday_is_friday: Was the article published on a Friday?

weekday_is_saturday: Was the article published on a Saturday?

weekday_is_sunday: Was the article published on a Sunday?

is_weekend: Was the article published on the weekend?

LDA_00: Closeness to LDA topic 0

LDA_01: Closeness to LDA topic 1

LDA_02: Closeness to LDA topic 2

LDA_03: Closeness to LDA topic 3

LDA_04: Closeness to LDA topic 4

global_subjectivity: Text subjectivity

global_sentiment_polarity: Text sentiment polarity

global_rate_positive_words: Rate of positive words in the content

global_rate_negative_words: Rate of negative words in the content

rate_positive_words: Rate of positive words among non-neutral tokens

rate_negative_words: Rate of negative words among non-neutral tokens

avg_positive_polarity: Avg. polarity of positive words

min_positive_polarity: Min. polarity of positive words

max_positive_polarity: Max. polarity of positive words

avg_negative_polarity: Avg. polarity of negative words

min_negative_polarity: Min. polarity of negative words

max_negative_polarity: Max. polarity of negative words

title_subjectivity: Title subjectivity

title_sentiment_polarity: Title polarity

abs_title_subjectivity: Absolute subjectivity level

abs_title_sentiment_polarity: Absolute polarity level

shares: Number of shares (target)

There are two non-predictive features, *url* and *timedelta*. They get to be removed right away. Before rescaling and taking a first shot at GD, note features like *weekday_is_nameoftheday* and its similars appear to be the reminisce of a once categorical feature that stored the day of the week (or weekend) in which the article was published. Another example are the features *data_channel_is_channelname*, with blogging categories. They were one hot encoded at some point, so no extra processing is needed here.

For the rest of the features, they seem to be arbitrary integers or decimals, so they were rescale as previously stated: a min-max scaler for all features as a starting point to preserve the day of the week column values, which are binary by means of their one hot encoding.

In respect to feature selection, I tried dropping features that seemed not to help at all. Dropping them usually implied an increase to the MAE metric, but as this increase never went past 10% relative to the MAE with all features, I decided not to drop them, but nevertheless I still tested both datasets (with and without them) when I increased model complexity.

I tried to substitute the min-max scaling with a standard normal scaling (ignoring or not the one hot encondings), but results were more or less the same (within the error from k-fold training). Because of that I decided to keep the min-max scaling, which was simpler.

When making histograms for all features, I noticed there were some outliers there, within feature distribution. Those might be either real outliers or just some measurement/calculation problem during dataset manufacturing. So a cutoff was calculated to keep 99% of the data and, one by one, select features that showed these outliers got them removed. This process removed almost 4500 data points and the MAE for this experiment went down on validation data.

This summed up with outlier removal from the target distribution (*shares*) left a little less than 85% of the data which was then used for training.

4. Gradient Descent Implementation

I opted for a hand-tailored Gradient Descent implementation to be able to judge its performance in each of its iterations in respect to my loss function and model complexity. I also used that to better tune the update of its learning rate.

GD was trained with MSE as the loss function and two strategies for the learning rate: a dampening factor that updated the training rate every iteration (settled up as 0.999 but generally speaking, it was always very near 1) and an extra step that divided the current learning rate by the number of the current iteration, but did not update the value for the next iteration like the dampening did.

5. Experiments and Discussion

Experiments for this work were done in a couple ways: splitting the training data for validation and not splitting and training the model with all available data. The first approach is used to extract all the data needed to tune the model, help with selecting features, etc. The second approach is used to train the final model, which is then tested against the test dataset.

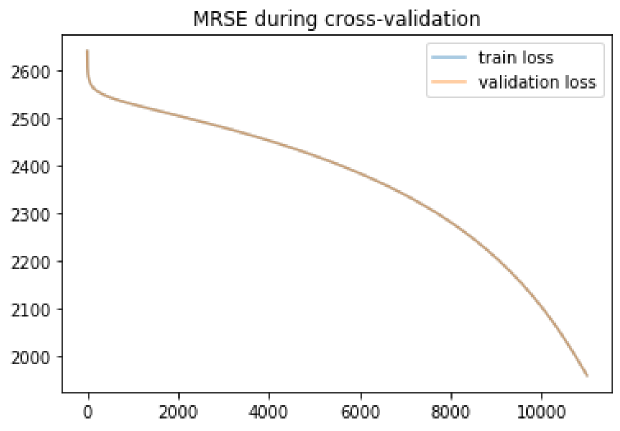
The full list of parameters follows, with decisions about data pre-processing and the optimizer. As it can be noted, no added complexity by polynomial features was used because it only seemed to help increase MAE. I did not train a model for each added feature, only by batches, and none decreased validation score.

Target outlier removal cutoff by the 95th percentile
Feature outlier removal cutoff by the 99th percentile
Scaling min-max
Iterations Up to 12000 for the final training, up to 10000 during crossvalidation and tuning
Learning rate Tested inside the interval $[10^{-}3, 10^{-}5]$ fixed and diminishing factors, decided for 10^{-3} with decay and normalization
Learning rate decay 0.999 (updates the learning rate each iteration
Learning rate normalization by the current iteration number (does not update the learning rate)
Regularization factor Tested inside the interval $[10^{-}5, 10^{-}8]$ and 0, decided for 10^{-7}

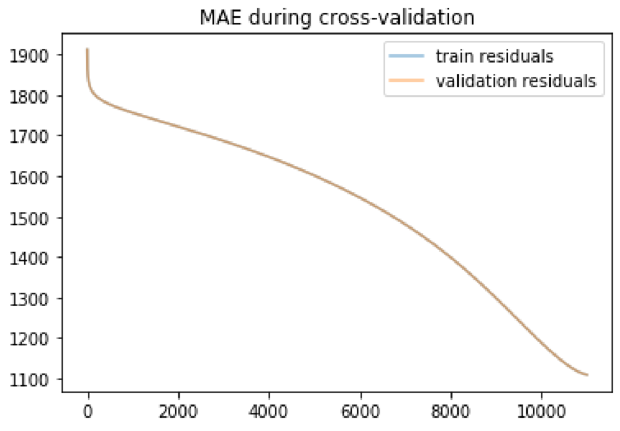
Following the initial dataset distribution, 80% train and 20% test, I decided to run a 5-fold cross-validation on the training set while tuning the model. The results after I settled up for the solution described in Section 3 with the aforementioned parameters can be seen in Figure 1 for both MSE and MAE during training. Note that there is only one visible line in both plots because they actually were ontop of each other.

As mentioned in Section 4, I opted for a ground up implementation of GD to control what could be taken from it while it iterated through the data. The result is GD returned not only the calculated coefficients and bias for the model, but also it's loss over each iteration for both the training and test sets. That was useful to produce a second plot, similar to the one of Figure 1, that can be seen in Figure 2 which shows averaged loss over iterations during cross-validation and loss over final model fitting for both train and test data, according to respective legends.

It only remains to say that the error margin during cross-validation was near 5 shares above or below predicted values while the test set made that figure go as high as 50, with the absolute difference between means of RMSE going up to 86 in the final model.



(a) RMSE loss over training

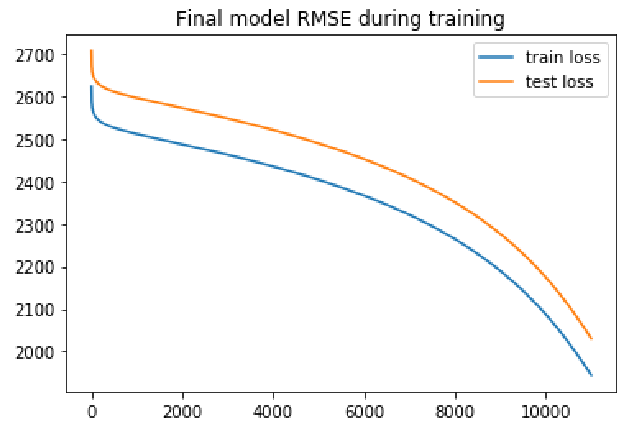


(b) MAE loss over training

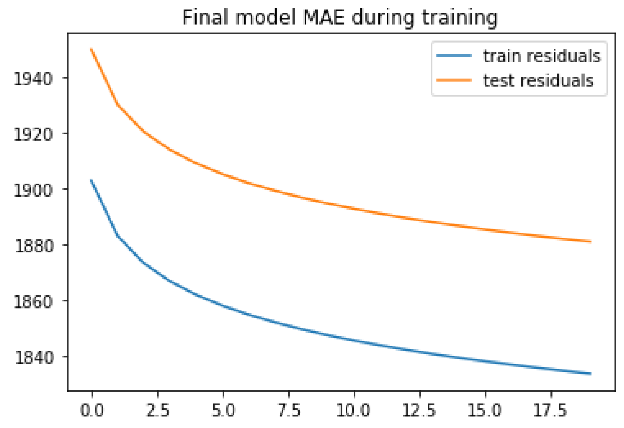
Figure 1: Cross-validation results

6. Conclusions and Future Work

Using a linear regression method against this dataset proved very challenging because not so much of the data features themselves, but more because of the simplicity of the model. The obtained fit was better than expected,



(a) RMSE loss over training



(b) MAE loss over training

Figure 2: Full dataset results

a result that gets very near the coefficients calculated by ordinary least squares (either by using the equation $w = \left(X^T X\right)^{-1} X^T y$ or linear regression solvers, which encapsulate the same behaviour), which are all within the same margin of MAE, calculated between 2500 and 2600. A more impressive feat comes from the baseline solution made without any treatment on the features, resulting in MAE as high as 15000.

Nonetheless, taking a step back, it must be also noted the predictions were all inside the 1100-1800 shares range, which also means the MAE was still high enough this might be a longshot from the true number of shares an article can attain. The main problem resides not only in the simplicity of the model, but on the features themselves. The dataset might not have the most desirable features to such a kind of prediction.

Finally, there are also the share outliers. These are viral articles that a model such as this cannot ever fully embrace, mainly because viral articles reside in a very different range of shares a hundred times greater than the rest of the data. And again, no feature appears to be capable of discriminating them without making the model less precise on the other side of the spectrum, where the other 99.85% of the data lives.

There are two obvious follow-ups to this study: first to employ better models than linear as it might be clear from the results so far, and second might be the use of an ensemble so not only one would be able to find the number of shares for a regular article, but also able to predict when an article will go viral and how many shares will it get.

References

[1] Kelwin Fernandes, Pedro Vinagre, and Paulo Cortez. A proactive intelligent decision support system for predicting the popularity of online news. In Francisco Pereira, Penousal Machado, Ernesto Costa, and Amílcar Cardoso, editors, *Progress in Artificial Intelligence*, pages 535–546, Cham, 2015. Springer International Publishing. 1