Machine Learning for Data Science HW3 Logistic Regression

Nikolay Kormushev

Introduction

In this report, I implement generalised versions of logistic regression, a multiclass variant known as multinomial regression and ordered regression which relies on an ordered distribution of labels.

I also explore interpretability of the weights of the multinomial model against a non-ordered dataset and I create a toy dataset on which an ordinal model performs better than a multinomial one.

Implementation Details

Both

For both models I add an intercept and use the $fmin\_l\_bfgs\_b$ optimizer with negative log likelihood as a loss.

Multinomial Regression

For multinomial regression I use a reference class. Due to this and the intercept it has $(m+1)\cdot(c-1)$ weights where m is the number of features and c the number of classes.

Ordinal Regression

Let our barriers for Ordinal Regression be θ_i. I set $\theta_0 = -\inf$, $\theta_n = \inf$ and since I add an intercept I can fix $\theta_0 = 0$ because it is offsets by the intercept to its correct position.

Instead of learning the barriers I learn their differences $\delta_i = \theta_{i+1} - \theta_i$ and use their cumulative sums as the barriers. I bounded $\delta_i > \varepsilon$ so the barriers are in increasing order.

The number of parameters is then is m for the feature weights and $b-2$ for the barriers so in total $m+b-2$ which is much less than the multinomial regression and results in the model converging faster and having lower variance.

Testing

For testing I used the python unittest library. To validate my methods I compared my resulting losses to well established libraries like scikit-learn and statsmodel configuring them to use a similar optimizers.

Most of my other tests were component tests for my regression model classes which looked at my methods in isolation to test for edge cases.

Results

Evaluation

I compared models based on their negative log likelihood. For accuracy I ran my tests 100 times using bootstrapping or generating new train and test sets.

Weight interpretation against basketball dataset

In this section I will look into interpretability of weights of a multinomial regression, using a Basketball dataset to determine what type of shot was made based on the distance and angle from the hoop, etc.

Choosing a reference class

For a reference class I chose *other* cause it encompases a lot of different types of shots which could make the weights harder to interpret. The reference class is not on the plot cause all of its weights are zero and can lead to a false sense that the features don't matter for it. Nevertheless the zero weights are used as a reference point for all other weights, negative meaning less and positive more than the reference class weight. Thus negative values mean in some sense a feature matters less than it does to the reference class.

Categorical variables

As part of preprocessing I one hot encode all categorical variables into separate features, of which I drop one. The reason for the drop is to improve interpretability. Without it each resulting feature is a linear combination of the others which means we have infinitely many solutions for the weights e.g. we know that a shot is either one or two legged. As a result we can get a high weight for *OneLegged* and a low one for *TwoLegged* creating the illusion that the second feature does not matter. This is not the case however because the model infers the information about it from $OneLegged = F$.

Normalising data and the intercept

During preprocessing I normalise numeric features. This helps because scaling the features down leads to the associated weights having higher values making them easier to interpret. Also it solves issues with sigmoid and softmax overflows.

During normalisation we remove the mean from the features which results in us having a higher intercept that returns the data to its original location. This is why its value is very low in Figure 1.

The intercept itself can also be seen as our prediction when all other weights or values are zero. This means it shows how likely is one class without any information and as a results follows the distribution of our data. This can be seen by looking at the sample sizes for each class: above head(3055), layup(973), hook shot(397), dunk(99) and tip-in(61).

Other(439) has an intercept of 0. Other intercepts values are relative to it and are negative because they have less samples than it. An exception is made by layup which has a lower intercept than ’other’ even though it has more samples. It is possible due to the ambiguity of ’other’ that the model figures out it is comprised of a lot of possible shots and so has a higher probability despite the dataset.

Heatmap results

Looking at Figure 1 we can see the weights for our features. Higher values are good increase the chance of predicting a class and low values a decrease it. These values can be positive or negative but we should not give too much weight to those signs because they are relative to the reference class.

An example of this is that with reference class other distance has a negative impact on dunk which makes sense since we are too far to dunk. But when my reference class was tip in this value was positive which seems illogical but when compared to other classes it was the lowest value. It was positive only because the tip-in weight was set to 0 and we wanted the distance to have less of a negative impact on the dunk comparatively.

Still we can sometimes have some nice interpreations like being in the under 14 competition means it is unlikely you can dunk so it has a negative weight. On the other hand the distance weight for an above head shot is positive because if you are far away from the hoop it is one of the few shot you can take.

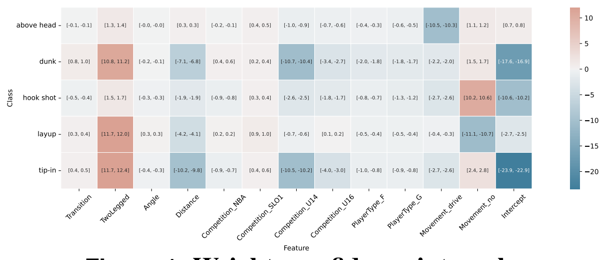

Figure 1. Weights confidence intervals

Comparison of Ordinal and Multinomial Regression

Toy dataset

To compare the models I made dataset with ordered classes and overlapping distributions to avoid infinitely many solutions due to perfect separability.

First I took a maximum value *max_val* and split the range from 0 to *max_val* into equal bins. These bins are my class labels. My dataset features are randomly generated integers. I take a sum of them, applying a non-linear function to one of them and got a value I will call $feat\_sum$.

To determine a class label we take $feat\_sum$ and check in which bin it falls. The class is the index of that bin. To make the distributions overlap if we are close to the edge of the next bin there is a percentage chance that we will be classified as being in the next bin.

Performance

In the last section created an ordered dataset. Because of this we would expect the Ordinal model to perform better. And from what we see in Figure 2 from the difference of the negative log likelihoods of both models we can conclude that it does it for small datasets.

The reason is that the ordinal model has less weights because it makes use of the ordering of the data. As a result the model converges faster because it has less variance. As we increase the train set size we decrease the variance of the weights of the multinomial and it stops overfitting and starts performing better although both models are doing good. With the increase of the train set the difference continues slowly increasing in favour of the multinomial model. I believe that depending on the complexity of the dataset the difference might become bigger since the greater amount of weights on the multinomial model means it can learn more complex patterns.

The evaluation was done on a test set of 1000 elements generated separately using the data generation process I described earlier.

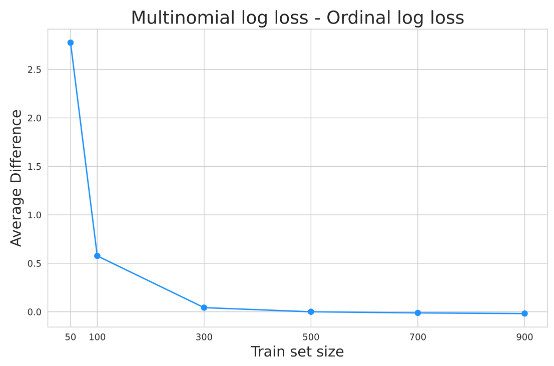

Figure 2. Loss difference when increasing sample size

Conclusion

In conclusion the multinomial regression is a model with interpretable weights but it is important to be careful when preprocessing and interpreting the data the because it is easy to be mislead.

We also saw how ordinal methods have good results on small train sets because they converge faster but because of the higher amount of parameters multinomial regression performs better when we have enough training data.