CS-433 Machine Learning - Classification of ordinal outcomes for the analysis of injury severity using machine learning methods

Hilda Abigél Horváth, Artur Andrzej Stefaniuk and Julian Paul Schnitzler
School of Computer and Communication Sciences, EPFL

***Abstract*—In this project, we performed ordinal classification in order to predict injury severity in car accidents. To compare different models we introduced 3 evaluation metrics: discrete classification metrics, GMPCA and QWK. We evaluated different data processing methods: standardization, age binning and upsampling. We evaluate different machine learning models: a basic FeedForward Net, added Weight Regularization, also a Poisson PMF layer, and different loss functions: MSE, MAE, cross-entropy and weighed cross-entropy and a ordinal approach. The FeedForward Network for Ordinal Classification using MSE and only standardization for pre-processing performed the best overall. We could achieve 87.13% test accuracy, 0.717 GMPCA and 0.783 QWK. The achieved accuracy is 3% better than the previous model from [1].**

I. Introduction

Traffic accident analysis is important for road safety and prevention. Traffic accident data is often collected in post-crash surveys. These surveys utilise ordinal data to assess the injury severity. The aim of this project is to create a machine learning model that predicts the expected severity of injuries sustained in a crash, given other information about the accident.

Classification of ordinal data is a special case of a classification problem, where the categories are naturally ordered. Contrary to the normal classification problem, the classes are correlated and not independently and identically distributed. They can be viewed as a scale that usually ranges from best to worst possibility (for example from no injury to severe injury). However, the distance between the categories is not known, so a simple regression model cannot be used to predict such outcomes. Moreover, when classifying ordinal data, the likelihood of the category to be chosen should reflect the natural ordering of the data. So, the second most probable outcome should be one of the categories that is adjacent to the most probable choice. To take into account the ordering of the data, specific regression models needed to be developed, such as Unimodal Logit [1].

This project aims to create a model based on a neural network, that can predict ordinal outcomes and to compare its performance to the Unimodal Logit model. The paper is structured as follows. Section II explores the data set used to train the model and describes the pre-processing methods that were used to improve the predictions. Section III discusses performance metrics used to evaluate the models, since due to the proprieties of the ordinal data, several metrics besides the accuracy needed to be used. The construction of the models and different loss functions used during the project is described in section IV and V respectively. Finally, the evaluation of all aforementioned parameters is presented in section VI.

II. Data Pre-processing

The data we use to develop the model is a publicly available data set of the High Severity Traffic Crash Data Report from the City of Tempe, Arizona[1]. Our task was to develop a model based on the processed data used in the Unimodal Logit paper [1][2].

This processed data consist of 39,793 records with characteristics of the crashes, mostly encoded as one-hot vectors. The data contains different aspects of the incident which can have an influence on the injury severity, such as the weather, or the age of the driver.

Based on these available data the model should estimate the injury severity on a scale from 1 to 5. The associated labels are *'no injury', 'possible injury', 'minor injury', 'major injury'* and *'fatal injury'* respectively.

Table I: Distribution of data between the injury severity categories in the crash data set

Injury severity	Number of data
'no injury'	27,473
'possible injury'	7,251
'minor injury'	4,411
'major injury'	559
'fatal injury'	99

As can be seen in the table II, our data set is really imbalanced, vast majority of records are from crashes with 'no injury'. This can make the model biased to estimate 'no injury' in more cases. To handle this and other problems we have used the following methods to further pre-process the data.

A. Standardization

We experienced that, in the data set, each characteristic of a crash is expressed as a binary variable except the age and the number people injured in the crash. We had to standardize these non-binary values. For standardization, we used the z-score:

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$$

Here x_{ij} denotes the previous value, μ_j and σ_j are mean and standard deviation of that respective characteristic, and z_{ij} the transformed x_{ij}.

B. Age

Some of the age values were missing or were denoted by 0. This can substantially distort the results, because for 4,622 records the model would consider that age of the person is zero. We solved this issue in two ways. First, we imputed the mean age values to replace 0 values. Then, we created an additional variable describing if the age was available.

[1]The raw data available here: https://data.tempe.gov/datasets/tempegov::1-08-crash-data-report-detail/about

[2]The processed data we used is available here: https://github.com/mwong009/unimodal-logit

It's value was set to 0 when we had data about the age and 1 otherwise.

C. Age binning

We also tried another method to handle data associated with age, that skipped the missing ages issue entirely.Based on the paper about the association of driver's age and the risk of causing a crash [2], implemented a bin structure as seen in the table II. This approach offered an alternative insight into driver age where the severity depends on whether someone is young or old but not on their exact age. Additionally, converting the age from continuous variable into categories created a data set with only categorical features.

Table II: Distribution of data between the different age groups

Age group	Number of data
18 - 24	12'088
25 - 34	8'603
35 - 44	4'761
45 - 54	3'820
55 - 64	2'876
65 - 74	1'540
75 +	432

D. Upsampling

As mentioned before, the data set used in training the model is not balanced. As it was not possible to change the initial data set, the problem of balancing the classes could not be solved by acquiring more data. Even if such an approach was possible, gathering more data would have likely not solved this issue since, fortunately, the vast majority of reported crashes result in no or minor injuries. Therefore, to deal with the issue of an unbalanced data set, an approach involving data processing needed to be used. Upsampling of data can offer a solution to this exact problem. In general, it consists of increasing the population of under-represented classes by sampling it more times than the over-represented classes.

1) Random upsampling: Random upsampling of data is the most straightforward way to implement upsampling. As such, it is considered a baseline to which other upsampling methods can be compared. In this approach, all of the initial data is taken and each class is considered independently. All of the under-represented classes, where the number of data points is smaller than that of the biggest class, are then sampled to take additional data points until the size of all classes is equal. This simple resampling of data has been shown to improve the classification accuracy on the minority class according to Chawla et al. [3]. However, since this method is reusing a small number of the same data points to upsample the under-represented classes, it can lead to problems with classifying new data. By reusing the same points, the classification regions for these smaller classes can become very small and specific.

2) SMOTE algorithms: In order to combat the issue of region specificity, more complex resampling algorithms were developed. SMOTE (Synthetic Minority Over-sampling Technique) [3] approach focuses on creating "synthetic" data points that are situated between existing points from the same class to populate the under-represented classes. New data points are generated from an existing point by taking one or two of its nearest neighbours. For continuous features, the value for a new synthetic point depends on the Euclidean distance between the points, while for categorical features the new value is the median value between the points. Points created in such a manner will mostly be distinct from the initial data set but will still lie close to the original data. Since the algorithms enforce that the new points lie between the existing ones, the classification region will be filled out but not become too specific. To upsample the Crash data set two SMOTE algorithms have been used. SMOTENC, which works on a data set with a mix of categorical and continuous features was used on the data where the age was kept as a continuous value. The second algorithm SMOTEN works for purely categorical features, it was used for a data set where the ages were binned. Both algorithms were implemented using the corresponding functions from imbalanced-learn python library[4].

III. Model performance evaluation

We used three different metrics to evaluate the mode, Discrete Classification Accuracy, Geometric Mean Probability of Correct Assignment (GMPCA) and Quadratic Weighted Kappa (QWK). These ones are the same metrics used in [1], in this way we can compare our results to the ones in the paper.

A. Discrete Classification Accuracy

The most standard way to measure the accuracy is to count the number of correct assignments (each case is assigned to the class with the highest probability) and divide by the number of predictions.

$$ACC = \frac{\sum_{i=1}^{N} \mathbf{1}_{\{i_n = x_n\}}}{N}$$

However, during our model development we experienced that we could achieve really good accuracy with assigning every case to the 'No injury' category. This was the result of the fact that the data set is over-represented by 'No injury' cases. This metric is sensitive to the skewness, and as we discussed before our data set is unbalanced in the sense that the vast majority of data belongs to the previously mentioned group.

B. Geometric Mean Probability of Correct Assignment

This metric is defined as follows.

$$ACC_{GMPCA} = \left(\prod_{n=1}^{N} \mathbf{P}(i_n | x_n) \right)^{\frac{1}{N}}$$

This measures the geometric average correctness of the predictions, which is more efficient in evaluation than the previous metric. This is due to the fact that this evaluation method takes into account the confidence, i.e. percentage in softmax, of the prediction. So if the prediction is very confident, it punishes less then in worse cases.

C. Quadratic Weighted Kappa

This metric takes into account that correct assignment can be made by chance as well.

$$ACC_{QWK} = \kappa = \frac{ACC - \mathbf{P}_e}{1 - \mathbf{P}_e},$$

where $\mathbf{P}_e$ denotes the probability that the prediction matches with the observation according to the data set. If the statistics' value is negative it means that it is worse than random assignment.

IV. Model

We used several approaches to compare how the models developed and choose the best one. For neural network models, PyTorch [5] was used. Most evaluation metrics are from scikit-learn [6]. For data processing, we used pandas [7] and numpy [8].

A. FeedForward Net for Multi-class Classification

As a baseline Neural Network, we chose an architecture with one input layer, one significantly larger fully-connected hidden layer, and one output layer. The size of the input layer matches the number of features, while the output layer matches the number of classes. We decided for a ReLu activation Function on the Hidden Layer. The output of the output layer is transformed through a softmax function to receive a probability distribution among the classes. We then consider the maximum value among the outputs as the predicted class. The model was constructed with hidden size of 300, learning rate of 0.005 with the AdamOptimizer. The maximum number of epoch was set to 150. For regularizing weights and to avoid overfitting, we also tried a L_2 weight regularization with a factor of 10^{-5}.

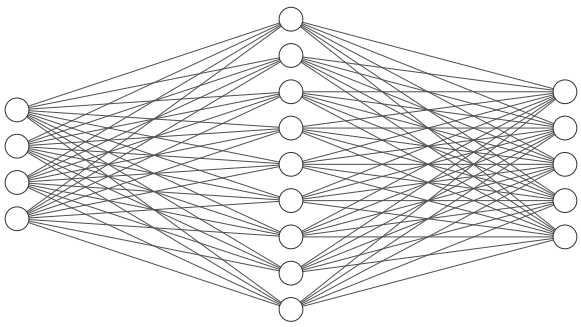

Figure 1: Sketch of our Neural Network architecture: In reality, the input layer has 33 nodes, the hidden layer 300, and the output layer 5 nodes

For training and testing, we transform the class-labels from our data set to one-hot encoded vectors in $\mathbb{R}^C$, with C denoting the number of classes, i.e. we map the class labels $0, 1, \ldots, C$ to vectors $[100\ldots0], [010\ldots0], \ldots [000\ldots1]$. This enables us to consider the problem as a multi-class classification problem.

B. FeedForward Net for Ordinal Classification

As presented in [9], for ordinal classes, it can be beneficial to use an ordinal encoding instead of the one-hot encoding presented in IV-A. We choose the same architecture as in Fig. 1, but map the class labels $0, 1, \ldots, C$ to vectors $[100\ldots0], [110\ldots0], \ldots [111\ldots1]$.
We take the output of the model into a sigmoid function, to map it strictly to a value between 0 and 1. We then get the class prediction by taking the maximum index for which the prediction is larger then 0.5. The advantage of this approach is the fact that it, when MSE is used, penalizes far miss-labels more than close ones, and therefore considers the ordinal nature of the classes.

C. Poisson PMF transformation

Another approach to consider is the transformation of the output into a Poisson PMF, which is presented in [10]. The authors argue that it can be beneficial for the performance to map a continuous regression output onto a discrete ordinal Poisson probability distribution.
We first take a 'copy' layer to copy the scalar output y into a vector $[y\ y\ \ldots\ y]$ of size C, and then apply the log of the Poisson probability mass function with $\lambda = y$, and take the logarithm. Taking the softplus function on y ensures that this is well-defined. We therefore get

$$\log(\frac{\lambda^k \exp(-\lambda)}{k!}) = k \cdot log(y) - y - log(k!)$$

for k between 1 and 5. This result is then added into a softmax layer to get values between 0 and 1.

V. Loss functions

The crucial part of training on data is to find proper loss function to minimize. In our work we tried the following approaches. Below N denotes the number of samples, y_i is the expected value and $\tilde{y_i}$ is the predicted value by the model. ($y_i, \tilde{y_i} \in \{1, ..., 5\}$.)
According to Gaudette and Japkowicz [11] using MSE and MAE loss functions, which are traditionally used on continuous functions for training, can also be applied to classification problems. Therefore, the performance of these functions was measured on our data set.
1) Mean Squared Error: This loss function is the classic MSE

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \tilde{y_1})^2.$$

2) Mean Absolute Error: The MAE has the formula

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \tilde{y_1}|.$$

3) Cross-entropy Loss: Cross-Entropy is a usual Loss Function used for Classifiaction

$$CE = \frac{\sum_{n=1}^{N} -\log \frac{exp(x_{n,c})}{\sum_{j=1}^{N}\exp(x_{n,j})} y_{n,c}}{N}$$

4) Weighted Cross-entropy Loss: This is the weighted version of the above function

$$CE = \frac{\sum_{n=1}^{N} -w_{y_n} \log \frac{exp(x_{n,c})}{\sum_{j=1}^{N}\exp(x_{n,j})} y_{n,c}}{N}$$

In our experiments, as our model predicted mostly classes 0 and 1, we gave a weight of 5 to class 2, 10 to class 4 and 20 to class 5, to avoid the overfit on overrepresented classes.

VI. Results

Previously discussed models and pre-processing methods were evaluated using performance metrics described in section III. Results are presented in tables III and IV. metrics that could not be calculated due to the nature of model were put as $*$. Confidence intervals in the tables are 10-fold cross-validation min and max values. We clearly beat our baseline, predicting everything to class 0, that resulted in an accuracy of 69%. We did not try more complex model architectures, since they did not show any promising improvement, given the imbalance in classes.

A. Standardization

Standardizing the input values did not improve the accuracy score but has improved all other metrics (see table III). Both GMPCA and QWK improved slightly. It seems that models trained on standardized and mean imputed values are more confident, probably due to the fact that the age values do not contain 0 outliers anymore.

Table III: Evaluation of the ML model trained on data that was pre-processed with different methods.

Model	Data processing method	DCA	GMPCA	QWK
Majority class	*	0.690	*	*
Baseline FeedForward Net	Simple	0.871 ± 0.017	0.717 ± 0.200	0.783 ± 0.027
	Standardization + Mean Impute	0.871 ± 0.012	0.765 ± 0.014	0.792 ± 0.029
	Binning Ages	0.871 ± 0.012	0.765 ± 0.014	0.792 ± 0.029
	Random Upsampling	0.825 ± 0.045	0.656 ± 0.255	0.697 ± 0.187
	SMOTENC Upsampling	0.773 ± 0.013	0.347 ± 0.004	0.590 ± 0.016
	SMOTEN Upsampling	0.827 ± 0.028	0.368 ± 0.007	0.824 ± 0.014

Table IV: Evaluation of the ML models trained with different loss functions.

Model	Loss function	DCA	GMPCA	QWK
Unimodal Logit[1]	*	0.839	0.581	0.758
Baseline FeedForward Net	MSE	0.871 ± 0.017	0.717 ± 0.200	0.783 ± 0.027
	MAE	0.868 ± 0.011	0.175 ± 0.053	0.749 ± 0.018
	Cross-Entropy	0.868 ± 0.010	$0.0 + 0.302$	0.752 ± 0.062
	Weighted Cross-Entropy	0.806 ± 0.007	0.0	0.851 ± 0.017
	Ordered Classes + MSE	0.870 ± 0.007	*	0.822 ± 0.017
Weight Regularization	MSE	0.867 ± 0.007	0.760 ± 0.050	0.813 ± 0.028
Poisson PMF	MSE	0.869 ± 0.008	0.638 ± 0.056	0.773 ± 0.022

B. Age binning

Our measures showed that for original (not upsampled) data the binning of ages did not change the performance of the model. We assume that due to the unbalanced number of data in the different bins the categorisation of this feature does not have any effect. For upsampled data, the binning of the ages increased the accuracy of the model by more than 5%. In this case, the difference probably comes from the performance of the two SMOTE algorithms. With only categorical features, it should be easier to create a more representative data set.

C. Upsampling

In order to evaluate the performance of the upsampling methods, the performace metrics were evaluated on the Baseline FeedForward Net model that was trained on the upsampled data. For the final evaluation the upsampling with SMOTEN algorithm was used as this methods showed better performance(see section VI-B). The value of accuracy was expected to drop since, by making the under-represented classes more significant, the model predict the most common class less. As shown in table III the accuracy score for the model using SMOTEN upsampled data was almost 5% worse than for the original data. Additionally, the accuracy using SMOTEN algorithm was comparable to that achieved with random upsampling. As such, the upsampling was not used for the construction of the final model. Nevertheless, it is important to notice that the model trained on the upsampled data set is capable of classifying the data into the two smallest categories: 'major injury' and 'fatal injury' (figure 2a), whereas before only the three biggest classes were considered (figure 2b).

D. Loss functions

With our final model we evaluated the different accuracy metrics, however all loss functions performed really well. These results can be seen in the table IV. The best accuracy 87.13% was achieved with MSE as the loss function. However, we also observed that using the ordinal loss yields far more predictions in the high classes, and in general distributes the predictions more than the other loss functions, while almost matching the accuracy of the normal MSE. Weighted Cross Entropy loses accuracy by also predicting the highest classes.

$$\begin{pmatrix} 5492 & 0 & 0 & 0 & 6 \\ 0 & 450 & 927 & 34 & 0 \\ 0 & 243 & 634 & 37 & 0 \\ 0 & 30 & 72 & 10 & 0 \\ 11 & 4 & 7 & 0 & 2 \end{pmatrix} \quad \begin{pmatrix} 5498 & 0 & 0 & 0 & 0 \\ 0 & 1391 & 20 & 0 & 0 \\ 0 & 889 & 25 & 0 & 0 \\ 0 & 102 & 10 & 0 & 0 \\ 13 & 11 & 0 & 0 & 0 \end{pmatrix}$$

(a) prediction on data upsampled using SMOTEN (b) prediction on the original data

Figure 2: Confusion matrices for predictions on test data

E. Model architecture

Regularizing the weights here makes us slightly worse in terms of Accuracy, while it increases both GMPCA and QWK, hence leading to a more sophisticated model. The Poisson PMF layer is also slightly worse than the baseline model. We assume that it performs better with deeper architectures, as presented in [10], which due to our dataset was not an option here.

VII. Discussion & Conclusion

In this project we tried several approaches to solve the ordinal classification problem by machine learning methods. While age binning and upsampling did offer a more robust model that was predicting all the classes, the performance of this model was worse that when no upsampling was used. The Baseline FeedForward Net model performed the best, all the loss functions gave comparable results and best accuracy, 87.13% was achieved using MSE. This model outperformed the previously published Unimodal Logit model [1] by 3%. An increase in accuracy already for a relatively simple neural network model shows promise that further improvement of this approach could lead to a very accurate model. Additionally, since upsampling with SMOTE algorithm can create a more robust model, a more complex neural network trained on a more balanced dataset might lead to even better results.

REFERENCES

[1] Melvin Wong, José Ángel Martín-Baos, and Michel Bierlaire. A unimodal ordered logit model for ranked choices. 2021.

[2] Karoline Gomes-Franco, Mario Rivera-Izquierdo, Luis Miguel Martín-delosReyes, Eladio Jiménez-Mejías, and Virginia Martínez-Ruiz. Explaining the association between driver's age and the risk of causing a road crash through mediation analysis. *International journal of environmental research and public health*, 17(23):9041, 2020.

[3] Nitesh Chawla, Kevin Bowyer, Lawrence Hall, and W. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Intell. Res. (JAIR)*, 16:321–357, 06 2002.

[4] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.

[5] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[6] 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*, 12:2825–2830, 2011.

[7] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[8] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[9] Jianlin Cheng, Zheng Wang, and Gianluca Pollastri. A neural network approach to ordinal regression. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, pages 1279–1284. IEEE, 2008.

[10] Christopher Beckham and Christopher Pal. Unimodal probability distributions for deep ordinal classification. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 411–419. PMLR, 06–11 Aug 2017.

[11] Lisa Gaudette and Nathalie Japkowicz. Evaluation methods for ordinal classification. In *Canadian conference on artificial intelligence*, pages 207–210. Springer, 2009.