Project Machine learning

Kamiel Temmerman

Universiteit Antwerpen

1 Introduction

Machine learning (ML) is a subfield within the landscape of Artificial Intelligence, focused on developing models and algorithms that let computers make well-informed predictions or decisions based on data. This approach makes machines process information but also leads them to autonomously learn patterns and trends(Raschka and Mirjalili, 2017). Within machine learning, neural networks play an important role. They are based on the structure of the human brain and are build using interconnected layers of artificial neurons. Each layer then contributes to the extraction and interpretation of features from input data (Nielsen, 2015). Using them improves the results of machine learning algorithms. They do this by navigating the data better leading to elevated problem-solving(Goodfellow et al., 2016).

In this short paper, the focus will be the on the development of a machine learning algorithm and a neural network for predicting the ratings of various alcoholic drinks.

2 Description of your system

2.1 Data

The dataset contained more than 2,000 reviews of various products, like alcoholic drinks, including wines, beers, and cocktails. Beyond those, the dataset also included reviews for various food items and products such as lip balm and chewables. The data was taken from Kaggle but it lacked comprehensive metadata. It appeared to have been compiled from an application that gave users the possibility to evaluate products with numerical ratings, recommendations, and accompanying written messages.

Column	Non-Null Count	Dtype
Brand	1003 non-null	object
Name	1060 non-null	object
Data	1060 non-null	object
Recommendation	554 non-null	object
Helpful	437 non-null	float64
Rating	1060 non-null	float64
Text	1060 non-null	object
Title	1029 non-null	object
Weight	512 non-null	object

Table 1: Info dataset

The dataset needed extensive preprocessing to filter out noise. This involved many steps, leading to a big reduction in the number of observations from 2889 to 1293. It underwent further reduction after the dropping of the 233 empty entries in the Rating column(table 1). This step was undertaken because replacing the missing values with the average of the column would be wrong given its discrete nature. The final step was assigning shorter labels to the column names for clarity and efficiency. The most important step was looking at the frequency distribution in the Rating column and this revealed a great imbalance (table 2).

5.0	79.8%
4.0	10.6%
3.0	4.3%
2.0	2.8%
1.0	2.3%

Table 2: Distribution rating column

2.2 Methods

2.2.1 Ml

Choosing a machine learning model was restricted due to the discrete characteristic of the Rating column. So the model became a Text Classification Pipeline with an SGDClassifier. The 'rating' column was the dependent variable (y), while the 'text' column served as the independent variable (x). The 'text' and 'title' columns were not combined, seeing that this extended the model's runtime without giving better results.

The biggest obstacle was the class imbalance. To fix it two options were explored, a RandomOverSampler and class weights. The first led the model to overfitting and was thus dropped. The latter became the better option, because it led to a more effective model. Weights were assigned to the five classes based on their frequency of occurrence.

In the final pipline a CountVectorizer was utilized, which converts raw text data into a numerical format. Then an MaxAbsScaler got applied to the transformed feature matrix. This ensured that each feature's values was scaled by the maximum absolute value in the entire dataset, leading to better convergence for the classifier. The final step was the SGDClassifier. Importantly, the class weight parameter was customized using a dictionary, which contained precomputed weights for the different classes. Then a parameter grid was employed to explore and fine-tune the hyperparameters of our classification pipeline. Before settling on this pipline, rigorous experimentation with alternative vectorization techniques and employing different classifiers, including RandomForestClassifier and LinearSVC, were done which did not yield significant improvements in the overall performance of the model.

2.2.2 Neural Network

For the neural netwok a a multiclass classification model to predict the ratings seemed the best option. The first step was combining the 'Text,' 'Brand,' and 'Title' columns. This gave a small improvement in the model's performance, without an increase in processing time. The combined column served as the independent variable (x), while the ratings remained the dependent variable (y).

The first step was introducing a Term Frequency-Inverse Document Frequency (TF-IDF) vectorization technique, to transform the textual data into a numerical format. To address the class imbalance within the dataset, we employed the Synthetic Minority Over-sampling Technique (SMOTE). This technique involves generating synthetic instances for the minority class to balance the class distribution. This did not really give amazing results. However, it did change the model so that it was not just answering a 5 out of 5 every time. This caused the overall accuracy to go down, but the weighted average to go up.

After that the multiclass classification neural network got developed. The model was implemented as a subclass of nn.Module and consisted of a single linear layer (nn.Linear) mapping input features to output features. The forward pass of the model was then defined, which simply passes the input tensor through the linear layer. The code then utilized the torch.nn.CrossEntropyLoss as the loss function for training the model. Additionally, it included a validation accuracy function (val acc) that used the argmax of the predicted values to compute accuracy. The model was trained using the utils.train function. This function took the model, loss function, validation metrics (cross-entropy loss and accuracy in this case), optimizer (Stochastic Gradient Descent with a learning rate of 0.1), training dataset (train iter), validation dataset (dev iter), and the number of epochs (100 in this case). The training history was then stored in the history variable, capturing relevant information such as training and validation loss, and accuracy over the course of the training process.

This first model's performance was not splendid, so the model had to be iteratively refined. This resulted in four versions, each with minor adjustments. Model 2 incorporated a deeper architecture with hidden layers, activation functions, and an EarlyStopper. In theory this would enable it to capture more complex patterns in the data compared to the simpler structure of Model 1. But the opposite seemed to be true. Model 3 was nearly identical in architecture, training configuration, and hyperparameters, the difference being that the Adam optimizer was used. In the last model dropout layers with a probability of 0.7 were used between the linear layers. This regularization technique randomly "drops out" a fraction of the neurons during training, preventing overfitting.

2.3 Results

2.3.1 Ml

During the evaluation of our machine learning model (table 3), we observed an overall accuracy of 75%, indicating that the model correctly predicted the target variable for the majority of instances in our dataset. Analyzing the precision scores, it was found that the model performed well in predicting instances belonging to class 5.0, achieving a high precision of 86% (see appendix, figure 2). Precision scores for classes 1.0, 2.0, 3.0, and 4.0 were notably lower, ranging from 9% to 28%. This means that predictions for these classes should be interpreted with caution, as the model tends to have a high rate of false positives. The highest recall (if a model captures all instances of a particular class) is observed for class 5.0, with a value of 90%, showing that the model effectively identified the majority of instances belonging to this class. However, lower recall values for classes 1.0, 2.0, 3.0, and 4.0 lead to many false negatives. The low weighted average f1-score of 74% indicates the overall balance between precision and recall across all classes.

	Precision	Recall	f1-score	Support
1.0	0.28	0.30	0.29	23
2.0	0.09	0.13	0.11	15
3.0	0.10	0.08	0.09	12
4.0	0.27	0.14	0.18	57
5.0	0.86	0.90	0.88	423
Accuracy			0.75	530
Macro avg	0.32	0.31	0.31	530
Weighted avg	0.73	0;75	0.74	530

Table 3: Classification report Ml model

So to sum up, while the model has a strong performance for the majority class (5.0), there is lots of room for improvement. Especially looking at the precision and recall for the minority classes (1.0, 2.0, 3.0, and 4.0). Developing other models and exploring additional techniques may enhance the model's ability to predict instances across all classes.

2.3.2 Neural Network

Because of the limited space, the focus will be on model 3 which gave the best results (see appendix, table 5). The outcome of the other models can be checked in the Google Colab file. The model architecture consisted of three linear layers with ReLU activation functions, and it was trained using the Adam optimizer with a learning rate of 0.001. The training process was 10 epochs (figure 1), and we employed early stopping with a patience of 5 epochs. The training history revealed a decrease in both the training and validation losses over the first epochs, indicating that the model was learning. However, after the fifth epoch, we observed an increase in the validation loss. This lead the early stopping mechanism to be activated, suggesting that the model started overfitting.

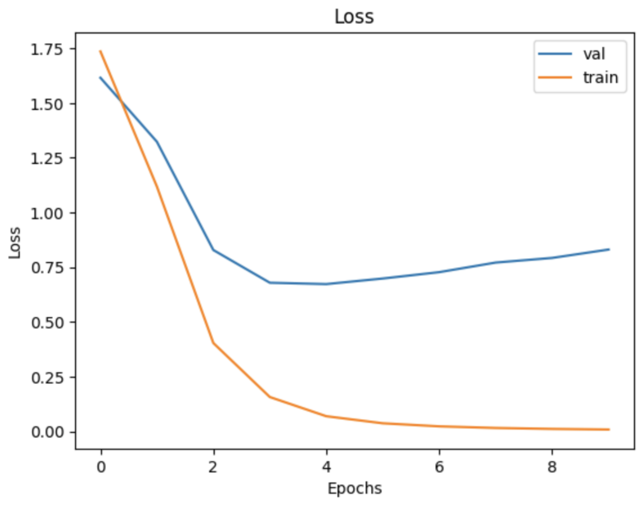

Figure 1: Training loss and validation classification, model 3

The evaluation on the test set gave reasonable results (table 4). The overall accuracy reached 79%, which is slightly better then the machine learning model but still bad considering the frequency distribution in the rating column. Precision, recall, and F1-score for individual classes was very mixed for the five classes. The model performed exceptionally well for the majority class (5.0), achieving high precision, recall, and F1-score values. For minority classes (1.0, 2.0, 3.0, and 4.0), the performance metrics are worse, indicating a failure in correctly predicting instances from these classes.

	Precision	Recall	f1-score	Support
1.0	0.67	0.18	0.29	11
2.0	0.50	0.12	0.20	8
3.0	1.00	0.21	0.29	6
4.0	0.38	0.21	0.27	28
5.0	0.82	0.894	0.88	212
Accuracy			0.79	265
Macro avg	0.67	0.33	0.38	265
Weighted avg	0.76	0.79	0.76	265

Table 4: Classification report model 3

3 Conclusion and Discussion

In this short paper we developed a machine learning and neural network model for predicting ratings of alcoholic beverages. They were based on a dataset containing various reviews for different products. It needed extensive preprocessing to end up with only the alcoholic beverages. The results were not spectacular. The machine learning model got an overall accuracy of 75%, with important variations in precision and recall across different classes. The neural network models, specifically Model 3, demonstrated reasonable accuracy, reaching 79%. But the same problem resurfaced here, resulting in success predicting the majority class (5.0) but failing to capture instances from the minority classes.

In conclusion, while the models presented in this paper show little promise in predicting all the ratings, there is lots of room for refinement, especially in handling the imbalanced class distributions. Future work could involve further fine-tuning of hyperparameters and experimentation with different solutions to correct for the class imbalance. The project, however, proved to be a valuable learning experience. Undertaking hands-on exploration served as an excellent introduction, providing new insights and resulting in a deeper understanding of the field.

4 References

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* (Ian Goodfellow, Yoshua Bengio, and Aaron Courville, Eds.) MIT Press.

Nielsen, M. (2015). *Neural Networks and Deep Learning.* San Francisco: Determination Press.

Raschka, S., & Mirjalili, V. (2017). *Python Machine Learning.* (2nd ed.). Packt Publishing.

5 Appendix

5.0.1 ChatGPT

ChatGPT was used for checking lines of code and helping me debug errors encountered during the process of developing the models. This was one of the first times trying to write some code for myself and GPT served as an invaluable tool to speed up the incredibly slow process.

5.0.2 Extra

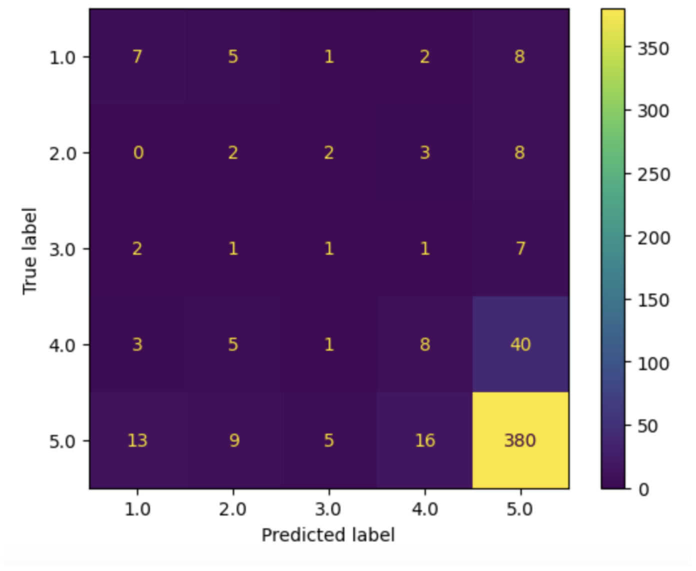

Figure 2: Confusion matrix display Ml model 3

	Accuracy	MSE	MAE
Model 1	0.735	1.196	0.479
Model 2	0.724	2.003	0.645
Model 3	0.792	1.071	0.4
Model 4	0.750	1.120	0.449

Table 5: Comparative results from the 4 neural network models