Dutch written review sentiment classification with a deep learning LSTM-model

Utrecht University

Master of Science in Applied Data Science

Supervisor: Rense Corten

Bas van Dalsum
9488561

July 1st 2022

Abstract

The online marketplace relies heavily on reviews to establish trust between the customer and the seller. These reviews are often both in written and star-rating form. In this paper, I use reviews scraped from the site werkspot.nl, a Dutch online market platform for small construction work, to analyse the sentiment reflected in the written review, and to predict the rated amount of stars. This study aims to investigate if the rating system can be improved by recommending star ratings based on the written review. I train a deep learning-based Bi-directional Short and Long Term Memory (LSTM) model to predict both the multi-class (5-star rating) and the binary sentiment (positive or negative) rating. The results suggest that this method has the potential to be of use for this application, but there are some issues to be resolved before this is possible. For instance, both models suffer from overfitting on the training data, resulting in lower model performance scores when testing the model on untrained data. One of the likely reasons for this is the extreme disparity between high and low ratings in the data. I elaborate more on this further in this paper. Overall, the main finding is that while the training results for both binary and multi-class are good, improvement is necessary to implement the model in real-world applications.

Contents

1 Introduction

The online marketplace is a hugely popular entity on the web, where many enjoy the convenience of being able to buy and sell whatever they desire. The site Werkspot.nl is a dutch marketplace where customers can post small-construction type jobs (e.g. plumbing, painting, flooring), and then tradespeople can respond by sending an offer. Then, the customer can see the reviews for the tradesperson, to aid the customer in deciding which offer to take up. The reviews of the plumbing section of the site will be used for the analysis in this paper, in an attempt to explore the relation between the written reviews and the star-based review scores. This is useful to aid in the betterment of the review system, as these reviews are the basis on which a customer decides to trust an offer or not.

In the online marketplace, there is a trade-off between trust and convenience which is different from offline transactions. In the traditional offline marketplace (i.e. shops) buyers and sellers are able to meet face-to-face during a transaction, which instils a level of trust and accountability between both parties which benefits the transaction. In contrast, the online marketplace offers varying levels of anonymity between the parties, ranging from transactions on the Dark Web where illegal substances may be sold and bought completely anonymously (Norbutas et al., 2020), to sites like e-bay.com where profiles are much more visible. How customers decide to make a transaction is commonly found to be a factor of trust and reputation (Bolton et al., 2004; Dellarocas, 2003; Teubner and Glaser, 2018). In the online marketplace, trust is especially important as buyers and sellers cannot be sure of each other's intentions and the quality of the product delivered. A potential customer's trust in a seller will determine whether or not they will make the purchase, as sellers with a higher trust score are perceived as more likely to honour the purchase agreement (Bolton et al., 2004; Norbutas et al., 2020).

The way this trust is determined is generally a combination of first and third-party experiences. Here first-party experiences are the dyadic relations, between the customer and the shop. This relation is formed by previous purchases., and directly influences the likelihood of a future purchase (Norbutas et al., 2020). Additionally, the third-party experiences are the reviews left by other customers, there is indirect reciprocity as the satisfied customer can positively others to purchase. Conversely, a dissatisfied customer can prevent others from making the same purchase by leaving a negative review (Bolton et al., 2004). The way this is often achieved in online market systems is a system of star ratings and a brief written review left by the customer.

The 5-star rating is a very useful system for getting an idea of whether a seller is trustworthy at a glance. These stars have an intrinsic meaning as most people will intuitively assume that 5 stars are the best and one star is the worst rating you can receive. There are however also some more well-laid-out definitions of the meaning of each star. Liu and colleagues, (Liu, 2017) propose that each star has a meaning beyond good or bad. In their paper, they propose that there is an emotional and rational rating system attached to the rating levels. They state that the 3 stars is a neutral rating, and 2 and 4 stars are

rationale-based ratings. Meaning that they convey more information on the attributes of the good or service. Whereas, the 1 and 5-star ratings are aimed to convey emotional qualities (Liu, 2017).

This way of looking at star ratings is useful, as it provides more information about the rated. But there are some caveats to the rating system as a whole. Firstly, not every customer leaves a rating. Approximately 63.9 per cent of customers leave reviews, and 72.6 per cent of customers read reviews (Cambria et al., 2017; Su and Shen, 2022). This complicates the research done on online review systems, as there is always a problem of missing data. Furthermore, in systems with both a star rating and a written review, there can be a level of discrepancy between the two. For instance, a 5-star rating with a complaint in the written text. This could diminish the effectiveness of the rating system as this causes the ratings to be less informative.

In addition, the reviews are almost never normally distributed. There is a wealth of research attempting to explain why this is the case (Teubner and Glaser, 2018). Generally, there is a large proportion of reviews that are 4 stars or higher, with lower ratings being far less common. In previous literature, there are many explanations given for why this happens. Filipas et al. (2017) propose that these high averages are the result of an interplay of factors. They found that review averages are likely influenced by the publicity and openness of the reviews, where review averages increase when the review is public. While the review was more candid, and lower scored, when the review was private (Filippas et al., 2017). Another reason for this skewness is found to be survivorship in the marketplace, which is highly influenced by negative ratings. As a negative rating is a direct predictor of exiting the market since offerings with negative reviews are less likely to be chosen by prospective customers (Teubner and Glaser, 2018).

Interestingly, this means that the system works well. However, there is a danger that the system works too well. When looking at the review scores in my data, and throughout the literature, it becomes clear that reviews that are not 5 stars are generally seen as negative (Bolton et al., 2004; Filippas et al., 2017). This causes the reviews to become less informative, as the average scores become increasingly high. Therefore, the average rating for a product or service starts to rise. Eventually, this reaches a point where the customer has a hard time again making an informed purchasing decision based on these scores.

Another factor to consider is that the focus of written reviews can differ vastly within the realm of product or service they are evaluating. This means that in the data we use there are reviews more focused ad hominem, describing more the plumber and his personal characteristics. But there are also reviews that are more descriptive of the quality of the product. Here I give two brief examples taken from the original dataset, with their English translations as the original reviews are in dutch.

> Ontzettend aardige man.[Really nice guy]
>
> Prima werk uitgevoerd en goede communicatie.[Good work delivered, and good communication]

1.1 Current study

Combined the problems of extreme skewness and the ratings being hard to interpret make the online rating system is flawed. The purpose of this paper is to increase the informational quality of the review scores, by trying to make the written reviews and the star rating more connected. I will make use of natural language processing techniques to process the written reviews of customers in order to make a model that is capable of generating a recommended number of stars based on the written review. I hope to achieve this by using a Long and Short Term Memory (LSTM) based deep learning approach in an attempt to predict the star ratings given based on the written text. Besides looking at the star rating I will also investigate the efficacy of this approach when trying to classify the text into positive or negative sentiment scores.

RQ1. Can I use a deep learning model to improve the informational quality of the star rating system by recommending a rating based on the users' written reviews?

RQ2. How does my LSTM classification model perform on both binary and multi-class sentiment classification?

RQ3. Is an LSTM model capable of doing text-based sentiment analysis on data with 5 labels?

2 Existing works

2.1 Sentiment analysis

Answering the research question will require my model to make use of sentiment analysis, which will allow the model to interpret written text into a rating score. Sentiment analysis is the task of analyzing people's sentiments taken from their written text data (Minaee et al., 2021). Minnaee et al. (2021) describe this as a classification task that often uses data with binary labels, usually positive or negative, for the classification (Minaee et al., 2021).

Most traditional methods of sentiment analysis are based on valence-aware dictionaries, which contain a score signifying the positive or negative quality of each word (Liu, 2017). Most sentiment analysis tasks are limited to polarity analysis and are not robust against things like negation or disjunction in texts (Liu, 2017). Liu (2017) describes several qualities of sentiment analysis that are important to the quality of the analysis when doing this kind of task. First, is sentiment orientation (positive, neutral, negative) which is the core scoring system. Then there is sentiment intensity (good vs. excellent), which can be determined by modifier words such as "very" or "dreadfully". Lastly, Liu (2017) names sentiment rating, a discrete rating system to represent the sentiment numerically or categorically (5 stars) (Liu, 2017).

This is related to my research question as often there are star-based systems in place next to the written reviews on platform marketplaces. But, these

are often seen as separate systems. Which can cause a disconnect between the sentiment reflected in the written review and the rated number of starts. In this paper, I aim to create a model with which a site or platform can recommend a number of stars based on the written review that is given. This will create a more connected system of reviews, which will likely increase the quality of information that can be gained from the rating systems.

2.2 Binary and multi-class classification

As stated in the previous section sentiment analysis is often done using binary labels. This is what studies like that of Shamrat et al (2021) use in order to determine whether tweets about COVID-19 vaccines had a positive or negative affect. They found that out of the examined tweets slightly less than half were classified as positive, with a smaller portion of the data points being classified as negative. Additionally, a small portion of the tweets was classified as neutral as they were not significantly positive or negative, according to the K-nearest neighbours classifier that was used (Shamrat et al., 2021).

Another example of binary classification is the paper by Shen and Su (2022) who looked at text analysis in order to determine the real sentiment of online product reviews. The caveat here is that the researchers did not only include real reviews, as they argue that with online reviews comes fraud. Meaning that the model they created was not online aimed at determining the sentiment, but also to filter out the fake reviews. Su and Shen (2022) achieved this using a Convolutional Attention Long and Short Term Memory model (CA-LSTM). I will elaborate more on these types of deep learning models in the next section. They found that using this model they were able to achieve an accuracy of 83.3 per cent, which was higher than all the models the authors compared it to (Su and Shen, 2022).

2.3 Deep learning models

In sentiment analysis, deep learning-based neural networks have been used to great success (Kim, 2014; Zhou et al., 2015), but there are many types of neural network architectures. But there are a few types of networks that are especially popular when it comes to using deep networks on classification tasks on written data. Two of these main architectures are the convolutional neural network (CNN) and the recurrent neural network (RNN). The CNN architecture makes use of pooling layers in order to capture correlations that allow the model to identify combinations of words in an input sentence. An example of this model is Kim et al. (2014) who in their paper make use of multiple filter layers in their CNN in order to allow the model to identify the relations of different words (Zhou et al., 2015). However, CNN might have issues when there are variable-length inputs and longer-term dependencies in the data. Here the RNN might be more suited, as the recurrency (information travelling back and forwards) in the model can allow the model to store and access historical information over time, meaning that there is an improvement over CNN models when the data

contains more dependencies over time-steps. This is useful when processing textual data, as the meaning of the full sentence is in itself a dependency (Zhou et al., 2015). However, the RNN starts to struggle when these dependencies as the time (distance) becomes greater. This is due to the fact that learning to store information over increasingly long periods of time takes up a lot of memory. Which is caused by insufficient decay of backwards error. In an attempt to address this issue Hochreiter and Schmidhuber (1997) proposed a novel architecture they named Long and Short-Term Memory (LSTM). The main difference here is that the decay of backwards error is dealt with by using special units that can retain information of data that has already passed through the model (Hochreiter and Schmidhuber, 1997).

2.4 Long and short term memory models

Tai et al. (2015) use various LSTM models in order to tackle two types of classification tasks on the Stanford Experimental Treebank database. This is a hugely popular database for classification tasks and is made up of over 11.000 fully parsed-out sentences. Completely labelled into a binary and a 5 class sentiment labels (Socher et al., 2013). Using these two sets of labels Tai et al. (2015) is capable of comparing the robustness of their model in classifying text sentiments into either positive/negative or a 5-label system. They do this in order to compare several models found in the literature, such as DCNNs (Kalchbrenner et al., 2014), CNN-Multichannel (Y. Chen, 2015). As well as some of their own models, which include: LSTM, bi-directional LSTM, and the Dependency Tree model. In their findings, they summarize that from all the models they tested the performance of the models on the multi-class (5-label) classification task the mean accuracy scores ranged from 43.2 to 51.0, with the highest scoring model being their own Constituency Tree-LSTM with tuned glove vectors. Reaching a mean accuracy of 51.0 per cent. This is quite good as it is higher than all the other models they tested it against (Tai et al., 2015).

However, this result means that there is still a huge discrepancy between the performance of classifying multi-class labels versus binary. This can easily be seen as the results of all the models Tai et al. (2015) tested greatly outperformed the best multi-class score when they were tested on the binary labels. In this task, the mean accuracy scores were in the range of 82.4 and 88.1. This time the best performing model was the CNN-Multichannel (Kim, 2014), with a 0.1 point lead over the authors' own best performing model (Tai et al., 2015).

3 Methods

In order to answer the research questions, I will take the following steps. First, I will have a look at the data. From here I will do some cleaning of the reviews where there are missing entries or other problems. Then, I will prepare the data for passing it through the model. After this, I can train the model on the cleaned data, and validate the results.

3.1 Data

The data I will use was scraped from www.werkspot.nl, a platform where the customer can post an advert for a job they need to do at their house. There are several categories, such as; painting, renovating and gardening. When the job is posted tradesmen are able to send their offers to the customer. Here is where the review system will play an important role, as the prospective customer is able to look at the accounts of the tradesmen and see their star ratings, as well as the written reviews they have received from previous customers. This is to help the customer to make an informed decision when choosing which tradesman to take up on their offer.

In this study, I use the data from a selected subsection of this platform, the plumbing section. The data has been anonymized to protect the customers' identities as well as the plumbers'. It contains the following; plumber id, date, star rating, the title of the review, written review, and customer id. In totality, there are 29.265 entries in the set, before cleaning. This is made up of 2267 unique plumbers and 19835 unique reviewers. There is some missingness in the data, with 726 entries without the star rating. In addition, there are 7213 entries where no written review has been entered. This comes to 24.6 per cent of customers who did not leave a written review. This is less than the averages found in the literature, where on average 36.7 per cent of customers do not leave a review (Cambria et al., 2017; Su and Shen, 2022).

In my data, the average star rating over all the entries is 4.63 stars, with the maximum possible rating being 5 stars. The skewness is -2.319, this is a very high level of right-sided (high rating) skew. This is to be expected, as this is a fundamental problem in the type of data that is being used.

3.2 Data preparation

3.2.1 Remove missing data

Data manipulation is a crucial step when dealing with written text data, in order to prepare the data for sure in my analysis. Here, I will give a short description of each step I take to prepare my data for the modelling phase, along with a short explanation of why I choose to do so.

First, I remove all the reviews where the rated amount of stars is missing. This is because upon further inspection these entries are all completely empty. I.e. all columns contain NA. Along with the reviews without star ratings, I also remove the entries where there was no written text left in the review. Although these rows in the data contain information on the rated number of stars, they are of no interest to my model. Which will be focused on processing the written text, in order to determine the number of stars that will be the most closely associated with the sentiment of the text. This is not possible when there is no text to process. This deletion makes the analysis easier, but also causes the loss of many points of data. Which is something to make note of. Additionally, the matter of what has to be done in the situation does not enter a written review

when leaving a new review will be addressed further in the discussion section of this paper.

3.2.2 Prepare strings

After all the missing entries are deleted, it is time to prepare the strings for feeding them into the model. First, all punctuation needs to be removed from the sentence, this is because these things have no intrinsic value in the model that I built. I do this using the 'punctuation' function of the 'string' package in python. This package contains a full list of all types of punctuation so that I can delete all items that are in this package.

The second step is to tokenize the string. Tokenization is the process of creating a separate string for each word in the sentence rather than having the whole sentence in one string. This is necessary in order to be able to analyze each word as a separate entry in the model. Without this, the model will analyse each entry as a full sentence, and will not be able to properly learn to assign the positive and negative weights to each word in the review.

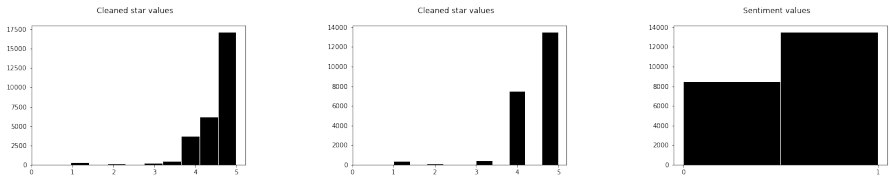
The next step I take is to remove all Dutch stop-words, which are words we use in our text and speech which do not have a meaning of themselves. The meaning of these words is generally fully dependent on the context surrounding them. Therefore, these words are not useful to attach sentiment values to, and should be removed. To do this I make use of the 'dutch-words' package. Which contains a list of Dutch stop words, such as; 'de', 'het' and 'om'. With this, I can delete all the tokens that are in this list. Creating a much more concise set of words, where each word is valuable to the model.

Review	Model-input	Stars	Sentiment
Goed, duidelijk en snel. Ik ben er blij mee	['', 'goed', 'duidelijk', 'snel', 'ben', 'blij', 'mee', '']	5	1
Na in eerste instantie uitstekend en snel geholpen te zijn met een storing	['', 'na', 'eerste', 'instantie', 'uitstekend', 'snel', 'geholpen', 'zijn', 'storing','']	2	0

Table 1: A snippet from the cleaned data, containing the original review, the review after the data cleaning and preparation, the star reviews and the sentiment score.

3.2.3 Stars

The last bit of data manipulation I perform is to reclassify the star ratings in two ways. In the collected data the customer was able to give whole and half stars in their reviews. However, this leads to having 10 classes in the data, as there are 10 possible ratings. Which, as I explained previously, is a lot to train



(a) All star ratings (b) Cleaned star ratings (c) Sentiment scores

Figure 1: Distributions of the rating scores before and after the data preparation. Figure 1a shows the complete data, before doing any cleaning or manipulations. Figure 1b shows the distribution of the star ratings after redividing them into whole-star only. Lastly, figure 1c shows the distribution of the reviews when they are rated as 0 (negative) or 1 (positive).

a classification model on. Therefore, I have decided to bring this back to 5 classes, one for each of the whole stars. I do this by rounding down the reviews to their previous whole, so 4.5 becomes a 4, 3.5 becomes a 3 etc. This achieves two things, one is that the model now has to train for classification on 5 labels, which will make it easier to fit the data. The second thing this achieves is that it reels in a little of the skewness in my data. As previously described, there is a very high level of skew in the data with a high percentage of the reviews being 4 stars or higher. This is possible problematic, as it is harder for the model to train on the lower ratings. Simply because there is a lack of data, which is exacerbated by the high number of high ratings. Knowing this, I decided to round the half-star ratings (i.e. 4.5) down. In an attempt to deal with some of this skewness. As this is likely to help my model with the performance.

The other way I reclassified is into binary labels, taking every star rating of 4 and under as 0 (negative) and the rest as 1 (positive). My reasoning here is that in the literature previously discussed, it has been shown that neural network classification performs better with binary labels, which brings me back to my second research question on the performance of the model for binary and multi-class classification. Doing my analysis on both the 5-class and the binary system allows me to compare the performance of my model side-by-side, which will provide me with some valuable information on how well the classification model performs.

3.3 Model

To achieve a model that is capable of natural language processing for sentiment analysis, I use a Bi-lateral Long-Short-Term Memory network with a convolutional layer (LSTM hereafter). I based this architecture on some of the previously discussed literature. Shen and Su (Su and Shen, 2022), have a similar network in their paper, with which they achieve good accuracy scores. Therefore, I will follow their network design, with some alterations based on other literature.

3.3.1 Architecture

The layers I use in my model are as follows;

- Embedding
- Dense
- Dropout
- Bi-Directional LSTM
- Dense

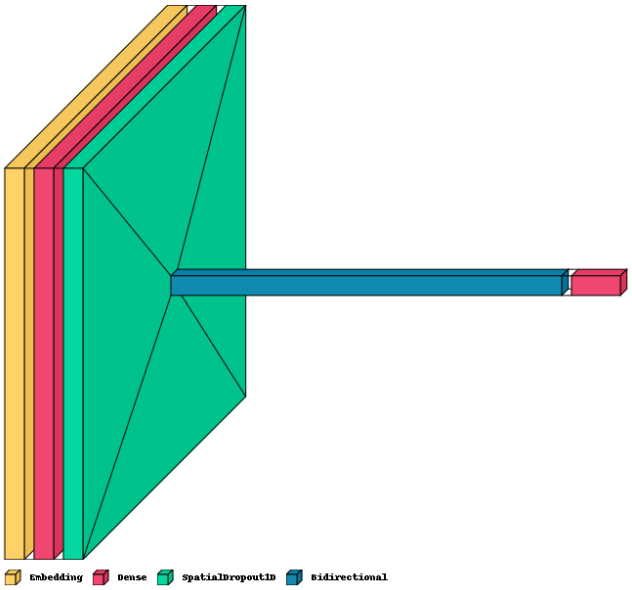


Figure 2: A visual representation of the model architecture. Starting with Embedding, then a Dense layer. Followed by a Dropout layer and then the Bidirectional LSTM layer. The last layer is a Dense layer again, to give the predicted output.

The first layer, Embedding, serves as a way to make all the inputs uniform. Here it makes all the arrays of the input sentences a length of 250 items, and it creates an embedding dimension matrix per word of 100 by 100. With this, the model can set weights for each of the 100 words that are the closest relations to the target word. Using this the model is able to identify combinations of words that have textual relations. The second layer is a fully-connected layer of 100 neurons this means that each of the 250 input layers is connected to each of the 100 neurons. Where the weights of each connection are altered during the training to create the most optimal activation mapping, this process creates

for each neuron a feature map, telling it to activate (or not) based on a certain threshold of inputs. The activation function I choose for this is 'Softmax'.

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K \tag{1}$$

The Softmax activation function, equation 1, is a common activation when dealing with multiple input structures. It works by taking as input a vector of real values and turning it into probabilities that sum to 1. Where large values get large probabilities and small values get small probabilities. Equation 1 shows the formula of the Softmax activation. Where each value in the input vector is divided by the sum of the full vector, to determine the probability of activation. This is useful for multi-class classification, as this function helps to divide the input layer into multiple classes (Su and Shen, 2022; Tai et al., 2015; Young et al., 2018.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

While softmax works quite well in the multi-class model it is important to note that this activation function does not function well in the binary model. In this model, I make use of the 'Sigmoid' activation. Which is a nonlinear continuous function, that outputs values between 0 and 1. This gives the probability that a node is active or not, meaning that the last layer is quite good at determining whether the review is positive or not.

The next layer is a one-dimensional spatial dropout layer (SpatialDropout1D) function in keras. The purpose of this layer is to randomly delete a set amount of data with each pass of the model. The amount of data to be dropped is determined by a set value p, for which I chose $p = 0.2$ This is done to prevent the model from becoming overly focused on features that are only relevant for select parts of the training data, i.e. overfitting. The reason I use the SpatialDropout1D and not the 'regular' dropout is that the SpatialDropout1D drops the entire feature maps instead of the individual element. This is important in convolutional networks as there is often a high correlation between the elements and simply dropping the element might not be enough to prevent overfitting (Tompson et al., 2015.

Following this layer is the Bi-Directional Long and Short Term Memory Layer (Bi-LSTM). This layer, adapted from various literature, among which: (T. Chen et al., 2017; Hochreiter and Schmidhuber, 1997; Su and Shen, 2022; Tai et al., 2015), will do most of its learning by taking the input reviews and identifying the structures that make up a negative or a positive review. This is done by using a hidden layer that preserves the inputs that have already passed it, in order to increase its learning efficiency. Here I choose to use a Bi-Directional form as, contrary to a Unidirectional one, the LSTM can use information from before and after the score, it is trying to predict. This is quite abstract, but conceptually it will look like the following. I will use an example provided by Kalchbrenner et al. (2014), where the target word is; 'on'. A unidirectional LSTM will take as information:

"The cat sat...

This leaves a lot of open room for the model to come up with possible interpretations, so we take the bi-directional approach. Which gives the following pieces of information:

"The cat sat..."

And it takes:

"... the red mat."

Using both the part of the sentence that comes before and what comes after makes predicting the target easier. As it makes use of the context around the target. The Bi-LSTM in my model functions similarly, by trying to assign sentiment values to a target, making use of the information that comes before and after the target value.

The output of this layer is sent to the final layer, which is a convolutional filter layer. This layer takes all the weights of the previous layer and uses an activation function to determine the activity of the nodes. This means that for the multi-class model the Softmax activation function is applied to calculate an activation probability for each of the 5 possible star labels. For the binary model, this process is slightly different. In this variant, the sigmoid activation function is applied to determine the probability of the final node becoming active (i.e. a positive rating).

3.3.2 Cross validation

To train this model I use the cross-validation method. Which is a training technique for model selection which helps to prevent overfitting by making use of a k-fold train-test split. This is done by splitting the data into train and test sets k number of times, to prevent the model from overfitting on a certain train set. And then for each split training the model, and measuring its accuracy using the corresponding test set. After this is done the scores of each fold are averaged into the final accuracy score. To train my models I use a k-fold validation of 10 folds, which is a commonly chosen amount in the literature (Minaee et al., 2021). In combination with 100 epochs of training per fold. To do this cross-validation I use 85% of the original data while reserving the other 15% for the final validation. To make sure that I test the model on data that it has not processed during training.

3.3.3 Hardware and software

The specifications of the hardware I used to run the modelling on is as follows;

- CPU: AMD Ryzen 5800H
- GPU: Nvidea GeForce RTX 3050ti Laptop
- RAM: 16GB at 4233mHz

3.4 Performance metrics

After constructing the models I trained them, using the cross-validation method described earlier. The metrics I used to evaluate the model were accuracy, loss, recall, precision and the F1-score. Here, I will give the formula and a brief description of each of these. Keep in mind that these scores all have a range of $[0, 1]$, with 1 being the maximum, as that means each predicted class is the correct class.

Accuracy is defined by taking the number of True Positives (TP) classifications (i.e. the classification is the same as the true class) and adding the number of True Negative (TN) classifications and dividing this by the total amount of classifications made. Both correct, and incorrect. This is useful for giving an overview of the models' performance, but it is hard to tell where the model could improve.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

One of the metrics that can give more information is precision, which is calculated by taking the number of True Positives and dividing that by the total number of positive classifications in the model. Therefore using both the true and the false positives. Therefore, the precision tells us the ratio of correct positive classifications in relation to the false negatives.

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

Closely related to precision is recall. The recall is calculated by taking the number of True Positives and dividing this by the number of True Positives added to the number of False Negatives. This metric gives information on the capabilities of the model to accurately classify positives as positive, as False Negatives should have been positive.

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

The last metric I use is the F1-score, which is the harmonic mean of both the recall and the precision. This measure is commonly used to give an overview of the classification capabilities of a model (Minaee et al., 2021). In the application of this paper, I will calculate the F1-score in two ways. For the multi-class model, I will use the weighted F1-score, as this accounts for the imbalance in the full review data. For the binary sentiment model, I will use the standard formula, as this is made to be used in binary classification.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{6}$$

4 Results

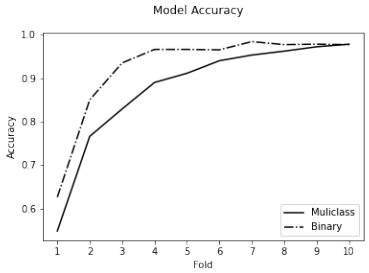
In this section, I will discuss the performance of both models based on their respective performance after training the model and using a previously unused part of the data for validation. The measures used to determine the performance of the models have been discussed previously.

4.1 Training

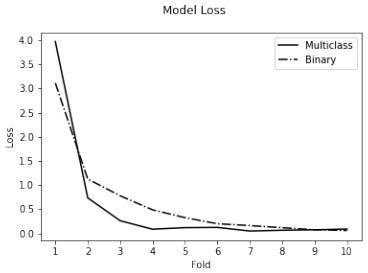
After the cross-validation is finished I have the scores of the performance metrics for each fold. In table 2 the averages of each metric are reported. The averages of these metrics are commonly reported to give a good idea of the overall performance of the trained models (Minaee et al., 2021; Su and Shen, 2022). The average accuracy and loss of the models for each fold in the training step are shown in Figure 3a and Figure 3b.

4.1.1 Accuracy and loss

Looking at Figure 3a There is a clear trend visible with the Binary model performing better from the start. Both with having a higher accuracy score and the speed with which the model improves. Of interest is that the Binary model stops improving around fold 4, with further folds providing only marginal improvements. This is different in the multi-class model, where the improvement is much more gradual. This means that the model took a longer time to improve, as the model took smaller steps. Interestingly, both models end up with similar accuracy scores at the end of the training. These scores can be found in Table 2, with the binary model scoring 0.92 and the multi-class model scoring 0.87.



(a) Accuracy scores over the 10-fold training period.



(b) Loss scores over the 10-fold training period.

Figure 3: Average testing accuracy and model loss of the 10-fold cross-validation training.

The inverse of this trend can be seen in Figure 3b, which shows the model loss over the folds. This score is a cumulative of the errors made by the classification, which means that a lower score is better Here, the Multi-class model starts with a higher loss score but drops more rapidly than the Binary model. Interestingly,

the loss scores of both models end up very similarly. With Table 2 showing that the Binary model has a slightly lower score; 0.56 versus 0.65 of the Multi-class model. Thus, the Binary model outperforms the Multi-class model here too.

4.1.2 Performance metrics

After looking at the average accuracy and loss of the model, it is time to look at the rest of the performance metrics. Looking at Table 2, it is evident that the Binary model consistently scores high on recall, precision, and F1-score. These scores are 0.94, 0.93, and 0.93 respectively, which means that the Binary classification model performs well with a few wrong classifications (both false positive and negative). These values provide evidence that this model is suited-well for the analysis of the overall sentiment in the text.

Measure	Binary	Multi-class
Accuracy	0.92	0.87
Loss	0.56	0.65
Recall	0.94	0.87
Precision	0.93	0.87
F1-score	0.94	0.84

Table 2: A table reporting the average scores of the folds in the cross-validation training step. These are computed by taking the model evaluation scores for each fold.

Furthermore, Table 2 also contains the average metrics for the Multi-class model. The average scores for this model are lower on all the metrics (smaller loss is better) than the scores for the Binary model. This is to be expected, as generally, the classification model will perform worse with each added class to classify. Regardless, the model performs well with a score of 0.87 on both the precision and recall and an F1-score of 0.84. In this situation, the F1-score is not in between precision and recall, because of the skewness in the data. In combination with the fact that this score is calculated for each class separately and then averaged. This way of calculating the F1-score gives a better, more complete, view of the models' actual performance (Minaee et al., 2021).

4.2 Validation

The model averages after the cross-validation training step give a very rosy picture of the performance of the classification capabilities for both models. But, these are likely too positive. In this I report the performance of the models on the validation data I set aside before the model training, making sure to use data that the model has not been trained on at all. These values are reported in Table 3.

As can be seen when comparing tables 2 and 3 there is a big difference in the scores for each reported performance metric. The recall for the binary

Measure	Binary	Multi-class
Recall	0.694	0.559
Precision	0.669	0.562
F1-score (binary)	0.681	–
F1-score (weighted)	–	0.559

Table 3: The final classification scores of my validation data, after the model was trained. The F1-score is reported separately as this measure was calculated with the weighted group sizes in the multi-class model. This makes it different from the normal form, and therefore not directly comparable.

model drops from 0.94 to 0.69, and the precision falls from 0.93 to 0.669. A similar trend is observed in the multi-class model. With recall going from 0.87 to 0.559, and precision declining from 0.86 to 0.562. This is a worrying trend, as this means that there is likely some degree of overfitting in my model training.

As discussed before; overfitting means that the model trains too much on attributes in the train data, and therefore it is not generalizable enough to use with new data. There are other explanations that might have contributed to the model performing worse on the validation set, but I will discuss these more in the discussion section. Before I do that I will conclude what these results mean for my research question in the next section.

5 Conclusion

In order to answer my research questions using my results, I will give a quick recap of the relevant results per research question. In order to look at the question of whether a deep learning model can be used to improve the informational qualities of the star rating system, I refer back to the results section. Here we see that indeed it is possible to train a deep learning model on the written reviews. Using this it is conceivable to build a system that asks the user to first leave a written review. Then input this review into the model, and let the model predict a number of stars. This prediction can then be displayed as a guideline for the customer so that they have an idea of what an appropriate amount of stars would be based on the review they wrote. Admittedly, this is all highly theoretical, and I will discuss some of the caveats in the discussion section. But, I would argue that if the model predicts the same number of stars for similar reviews then it will be successful in adding to the informational qualities of the star-rating system.

However, this is more of a theoretical answer. The answer differs somewhat when looking back on the results of the model tested in this thesis. As the validation section shows that there are more steps needed to achieve a model that is fully capable of implementation in this capacity. Looking back at the results in Table 3, I must conclude that there are still some issues to work out. Not least of which is the large difference in the scores reported in the validation

and after the training stage. Effectively, these differences mean that the trained model is not generalizable. In other words, this means that the model will not be capable of handing written text that is very different from the reviews that are present in the data as of now.

This leads us to the second research question. How does the proposed LSTM model perform on binary and multi-class sentiment classification? As reported in the results section, both models scored high in the training stage but lost performance in the validation. However, it becomes clear that the binary model scores higher on every metric in the training, and loses less in validation. Therefore, I argue that while both models are capable of doing sentiment analysis, the model performs better when the labels are either positive or negative. This is in line with the reviewed literature, which consistently finds that classification performance is better for binary labels than it is for multi-label problems (Kim, 2014; Shamrat et al., 2021; Su and Shen, 2022). This does not mean that the proposed model should not be used for the classification of star ratings, as the nature of this problem is a multi-class problem. But, it does mean that further optimizations are needed in order to improve the classification capability of the model. One way that would likely increase the model performance is having more balanced data, in particular having the data be more equally divided over the classes.

With that, the final research question can be answered. In the results section, it has been shown that indeed the LSTM network can be used to classify text-based sentiments into 5 labels, where each label represents a star rating. Especially reviewing the performance of the model after the training there is strong evidence that the LSTM model is capable of performing well in this task. Looking back on the results from Table 2, the model scores well on all the performance metrics. The scores of the model on the precision (0.87), recall (0.87) and the F1-score (0.84) are all close to 1.00, which is the maximum score for each of these metrics. Based on this, I conclude that an LSTM-based model has the potential to be of use in a text-based classification task.

6 Discussion

Overall, the performance of my model was quite good. But, when comparing it with other models, such as the CA-LSTM (Su and Shen, 2022), it becomes clear that it could have been better. It is however hard to compare the scores of my model with those found in the literature because all the other models are not trained on my data. In fact, many studies on deep learning-based approaches to sentiment analysis use the Stanford Sentiment Treebank. Which makes these papers convenient to compare to one another, but hard for me to compare my model too. The solution to this seems straightforward enough, either use the models from the literature and train them using my data. Unfortunately, this is infeasible within the scope of this thesis, but perhaps this will be an interesting follow-up paper.

Another point of interest is the skewness in the data. As this may have had

an effect on the training results of the data. The problem with this data is that the extremely high level of high ratings versus the very low level of low ratings makes it so that balancing the data is virtually impossible. This is easy with a curated data set, but with real-world data, this is a problem. Of course, I have already talked about this previously, as this was one of the main problems I was trying to improve upon in this thesis. In review data, there will always be some level of skewness, as this is the point of the review system where quality is rewarded

Lastly, another issue with the data I use is that there many reviews with no written text. Which, as previously described, I decided to remove from the data set as there is no text to train the model with. Nor is there text to classify. However, in real-world applications, a system must be in place to deal with customers who do not wish to leave a written review. One option to deal with this is to recommend a 3-star rating to all users without a written review. This is in line with the idea that if a user does not want to leave a review they are likely neutral about the product or service delivered (Liu, 2017). Alternatively, the recommendation could also be the current mean stars of the product or plumber in my case. These options could conceivably help with combating skewness, as the customer who gets recommended to give 3 stars might be less inclined to go for the 5 stars, but this is highly speculative and would be an interesting topic for further research.

7 Bibliography

References

Bolton, G. E., Katok, E., & Ockenfels, A. (2004). How effective are electronic reputation mechanisms? an experimental investigation. *Management science*, *50*(11), 1587–1602.

Cambria, E., Poria, S., Gelbukh, A., & Thelwall, M. (2017). Sentiment analysis is a big suitcase. *IEEE Intelligent Systems*, *32*, 74–80. https://doi.org/10.1109/MIS.2017.4531228

Chen, T., Xu, R., He, Y., & Wang, X. (2017). Improving sentiment analysis via sentence type classification using bilstm-crf and cnn. *Expert Systems with Applications*, *72*, 221–230.

Chen, Y. (2015). *Convolutional neural network for sentence classification* (Master's thesis). University of Waterloo.

Dellarocas, C. (2003). The digitization of word of mouth: Promise and challenges of online feedback mechanisms. *Management science*, *49*(10), 1407–1424.

Filippas, A., Horton, J. J., & Golden, J. (2017). Reputation in the long-run. *Available at SSRN 3103688*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1746–1751. https://doi.org/10.3115/v1/D14-1181

Liu, B. (2017). Many facets of sentiment analysis. *A practical guide to sentiment analysis* (pp. 11–39). Springer.

Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., & Gao, J. (2021). Deep learning–based text classification: A comprehensive review. *ACM Computing Surveys (CSUR)*, *54*(3), 1–40.

Norbutas, L., Ruiter, S., & Corten, R. (2020). Believe it when you see it: Dyadic embeddedness and reputation effects on trust in cryptomarkets for illegal drugs. *Social Networks*, *63*, 150–161.

Shamrat, F., Chakraborty, S., Imran, M., Muna, J. N., Billah, M. M., Das, P., & Rahman, O. (2021). Sentiment analysis on twitter tweets about covid-19 vaccines using nlp and supervised knn classification algorithm. *Indones. J. Electr. Eng. Comput. Sci*, *23*(1).

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. *Proceedings of the 2013 conference on empirical methods in natural language processing*, 1631–1642.

Su, Y., & Shen, Y. (2022). A deep learning-based sentiment classification msentiment classification real online consumption. *Frontiers in Psychology*, 1705.

Tai, K. S., Socher, R., & Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

Teubner, T., & Glaser, F. (2018). Up or out—the dynamics of star rating scores on airbnb.

Tompson, J., Goroshin, R., Jain, A., LeCun, Y., & Bregler, C. (2015). Efficient object localization using convolutional networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 648–656.

Young, T., Hazarika, D., Poria, S., & Cambria, E. (2018). Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, *13*(3), 55–75.

Zhou, C., Sun, C., Liu, Z., & Lau, F. (2015). A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*.

8 Appendices

A Data preparation

```
# Import the necessary modules
import pandas as pd
import string
import re
import stopwords


def main():
    # Import the data
    m_cols = ['Plumber_ID', 'Stars', 'Title', 'Review', 'Date',
    ↪  'Reviewer_ID']
    df = pd.read_csv('loodgieter_reviews.csv', sep=',',
    ↪  names=m_cols, skiprows=1, encoding='ISO-8859-1')

    # Transformations
    df.dropna(subset=["Stars"], inplace=True)  # Drop rows where
    ↪  Stars is missing
    # Delete or not? Need literature
    df.dropna(subset=["Review"], inplace=True)  # Drop rows where
    ↪  Review is missing

    df['Reviewer_ID'].replace(to_replace='Anoniem',  # Translate
    ↪  the Dutch word 'Anoniem' English
                              value='Anonymous', inplace=True)

    # call the function to prepare the strings
    df_prepped = prepare_strings(df, "Review")
    print(df_prepped.head())

    # Create a clean dataset
    df_clean = df_prepped[['Date', 'Stars', 'Review',
    ↪  'rev_no_punc_split_stop']]
    df_clean.rename(columns={'rev_no_punc_split_stop':
    ↪  'Review_Clean'}, inplace=True)  #
    ↪  .reset_index(inplace=True)

    # Add a binary sentiment label that is positve or negative
    ↪  with 5 stars being 1 (positive) and all else being 0 (
    # negative)
    df_clean['Sentiments_Binary'] =
    ↪  df_clean['Stars'].apply(lambda x: 0 if x <= 4 else 1)
```

```
    print(df_clean['Review_Clean'][4])

    # Save the cleaned dataset as plumber_clean
    df_clean.to_csv('plumber_clean.csv')


def prepare_strings(data_frame, target_column):
    # Step 0: initialize the data
    df = data_frame
    column = str(target_column)

    # Step 1: remove all punctuation
    df['rev_no_punc'] = df[column].apply(lambda x:
    ↪  remove_punctuation(x))

    # Step 2: Tokenize and convert into lowercase
    df['rev_no_punc_split'] = df['rev_no_punc'].apply(lambda x:
    ↪  tokenization(x.lower()))

    # Step 3: Remove stopwords
    df['rev_no_punc_split_stop'] =
    ↪  df['rev_no_punc_split'].apply(lambda x:
    ↪  remove_stopwords(x))

    # Step 5: reclassify the stars to 5 labels
    df['Stars'] = df['Stars'].apply(lambda x: round_stars(x))

    return df


def remove_punctuation(text):  # Function to remove all
↪  punctuation in a string
    no_punct = [words for words in text if words not in
    ↪  string.punctuation]
    words_wo_punct = ''.join(no_punct)
    return words_wo_punct


def tokenization(text):  # Function to tokenize the words
    split = re.split("\W+", text)
    return split


def remove_stopwords(text):     # Function to remove the
↪  stopwords
    # nltk.download() Use this to download the stopwords package
```

```
71      stop_words = stopwords.get_stopwords("dutch")
72      text = [word for word in text if word not in stop_words]
73      return text
74
75
76  def round_stars(star):  # Function to round down the star-ratings
77      x = star
78      if x <= 1.5:
79          star = 1
80          return star
81      elif x in [2.5, 2.0]:
82          star = 2
83          return star
84      elif x in [3.5, 3.0]:
85          star = 3
86          return star
87      elif x in [4.5, 4.0]:
88          star = 4
89          return star
90      else:
91          star = 5
92          return star
93
94
95  if __name__ == "__main__":
96      main()
```

B Models

```
# Import the modules

import numpy as np
import pandas as pd
import tensorflow as tf
import tensorflow_addons as tfa
import visualkeras
from keras.utils import np_utils
from keras_preprocessing.sequence import pad_sequences
from keras_preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Dense, Embedding, SpatialDropout1D,
↪  CuDNNLSTM
from sklearn.metrics import recall_score, precision_score,
↪  f1_score
from sklearn.model_selection import KFold, train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.metrics import Precision, Recall, categorical_accuracy


def main():
    # Import the data
    df = pd.read_csv('plumber_clean.csv')

    physical_device = tf.config.list_physical_devices("GPU")
    tf.config.experimental.set_memory_growth(physical_device[0],
    ↪  True)

    # Call the functions
    # Get the data, and save to the required variables
    x, y_sentiment, y_stars = get_data(df)
    # Get the multi-class model
    model_stars = get_model_multiclass(x)
    # Get the binary model
    model_sentiments = get_model_binary(x)
    # Use the data and the mode to train and validate the model.
    stars_recall, stars_precision, stars_f1_score =
    ↪  cross_validation(model_stars, x, y_stars, 10, 'stars')
    senti_recall, senti_precision, senti_f1_score =
    ↪  cross_validation(model_sentiments, x, y_sentiment, 10,
    ↪  'sentiments')

    # Print the performance metrics of the runs
    print('_____')
```

```
39      print(f'The validation values for the multiclass model
        ↪  are:\n'
40            f'Recall: {stars_recall}\n'
41            f'Precision: {stars_precision}\n'
42            f'F1 score: {stars_f1_score}')
43      print('_____________________________________________')
44      print(f'The validation values for the sentiment class model
        ↪  are:\n'
45            f'Recall: {senti_recall}\n'
46            f'Precision: {senti_precision}\n'
47            f'F1 score: {senti_f1_score}')
48      print('_____________________________________________')
49
50
51  def get_data(data):
52      df = data
53      max_words = 50000
54      max_seq_len = 250
55
56      # Tokenize the strings
57      tokenizer = Tokenizer(num_words=max_words)
58      tokenizer.fit_on_texts(df['Review_Clean'].values)
59      word_index = tokenizer.word_index
60      print('Found %s unique words.' % len(word_index))
61
62      # Truncate and pad sequences to make input all the same
        ↪  lengths
63      X = tokenizer.texts_to_sequences(df['Review_Clean'].values)
64      X = pad_sequences(X, maxlen=max_seq_len)
65      print('Tensor data shape: ', X.shape)
66
67      # Convert the star values to label variables
68      encoder_stars = LabelEncoder()
69      encoder_stars.fit(df['Stars'])
70      encoded_y_stars = encoder_stars.transform(df['Stars'])
71      y_stars = np_utils.to_categorical(encoded_y_stars)
72
73      # Covert the sentiment variables
74      encoder_sent = LabelEncoder()
75      encoder_sent.fit(df["Sentiments_Binary"])
76      y_sentiment = encoder_sent.transform(df['Sentiments_Binary'])
77
78      return [X, y_sentiment, y_stars]
79
80
81  def get_model_multiclass(x):
```

```
    max_words = 50000
    embed_dim = 100
    X = x
    f1score = tfa.metrics.F1Score(average='weighted', num_classes=5)

    # Construct the model
    model_multi = Sequential()
    model_multi.add(Embedding(max_words, embed_dim, input_length=X.shape[1]))
    model_multi.add(Dense(100, "softmax"))
    model_multi.add(SpatialDropout1D(0.2))
    model_binary.add(tf.keras.layers.Bidirectional(CuDNNLSTM(100,
                                                   recurrent_initializer='orthogonal',
                                                   kernel_initializer='glorot_uniform',
                                                   bias_initializer='zeros')))
    model_multi.add(Dense(5, activation='softmax'))

    model_multi.summary()

    # Compile the model
    model_multi.compile(loss='categorical_crossentropy', optimizer='adam', metrics=["accuracy",
                            Precision(),
                            Recall(),
                            f1score,
                            categorical_accuracy])
    # Save the visualization
    visualkeras.layered_view(model_multi, to_file='Multiclass_model.png', legend=True, max_xy=500, scale_z=10,
                             draw_funnel=True, spacing=10)

    return model_multi


def get_model_binary(x):
    max_words = 50000
    embed_dim = 100
    X = x
    # Function to calculate the F1-score
    f1score = tfa.metrics.F1Score(average='none', num_classes=1)

    # Construct the model
    model_binary = Sequential()
```

```
120         model_binary.add(Embedding(max_words, embed_dim,
            ↪  input_length=X.shape[1]))
121         model_binary.add(Dense(100, "relu"))
122         model_binary.add(SpatialDropout1D(0.2))
123         model_binary.add(tf.keras.layers.Bidirectional(CuDNNLSTM(100,
124
                                                    ↪  recurrent_initializer='orthogonal',
                                                    ↪  kernel_initializer='glorot_uniform',
                                                    ↪  bias_initializer='zeros')))
125         model_binary.add(Dense(1, activation='sigmoid'))
126
127         model_binary.summary()
128
129         # Compile the model
130         model_binary.compile(loss='binary_crossentropy',
            ↪  optimizer='adam', metrics=["accuracy",
131                                       Precision(),
132                                       Recall(),
133                                       f1score,
134                                       tf.keras.metrics.BinaryAccuracy()])
135
136         # Save the visualization
137         visualkeras.layered_view(model_binary,
            ↪  to_file='Binary_model.png', legend=True, max_xy=500,
            ↪  scale_z=10,
138                                  draw_funnel=True, spacing=10)
139
140         return model_binary
141
142
143     def cross_validation(model, X, y, k_folds=10, model_name=""):
144         # Get and set the variables
145         # Split the data into train and validation
146         X_train, X_val, y_train, y_val = train_test_split(X, y,
147
                                                          ↪  test_size=0.15,
                                                          ↪  random_state=111,
                                                          ↪  stratify=y)
148         inputs = X_train
149         targets = y_train
150         batch_size = 64
151         epochs = 100
152         verbosity = 1
153
154         # Define the K-fold Cross Validator
155         kfold = KFold(n_splits=k_folds, shuffle=True)
```

```
156
157     # K-fold Cross Validation model evaluation
158     # Define per-fold score containers
159     acc_per_fold = []
160     loss_per_fold = []
161     pre_per_fold = []
162     rec_per_fold = []
163     f1_per_fold = []
164     cat_bin_acc_fold = []
165     data_dict = {"Accuracy": acc_per_fold,
166                  "Loss": loss_per_fold,
167                  "Precision": pre_per_fold,
168                  "Recall": rec_per_fold,
169                  "F1 Score": f1_per_fold,
170                  "Categorical_Binary accuracy": cat_bin_acc_fold}
171
172     fold_no = 1 # Set fold number to 1
173     # Begin cross-validation
174     for train, test in kfold.split(inputs, targets):
175         model = model
176
177         # Generate a print
178
        ↪  print('------------------------------------------------------------------------
179         print(f'Training for fold {fold_no} ...')
180
181         # Fit data to model
182         history = model.fit(inputs[train], targets[train],
183                             batch_size=batch_size,
184                             epochs=epochs,
185                             verbose=verbosity)
186
187         # Generate generalization metrics
188         scores = model.evaluate(inputs[test], targets[test],
            ↪  verbose=0)
189         print(
190             f'Score for fold {fold_no}: {model.metrics_names[0]}
                ↪  of {scores[0]}; {model.metrics_names[1]} of
                ↪  {scores[1] * 100}%')
191
192         loss_per_fold.append(round(scores[0], 3))
193         acc_per_fold.append(round(scores[1], 3))
194         pre_per_fold.append(round(scores[2], 3))
195         rec_per_fold.append(round(scores[3], 3))
196         f1_per_fold.append(round(scores[4], 3))
197         cat_bin_acc_fold.append(round(scores[5], 3))
```

```
198
199             # Increase fold number
200             fold_no = fold_no + 1
201
202         # == Provide average scores ==
203
        ↪  print('------------------------------------------------------------------------')
204         print('Score per fold')
205         for i in range(0, len(acc_per_fold)):
206
            ↪  print('------------------------------------------------------------------------')
207             print(f'> Fold {i + 1} - Loss: {loss_per_fold[i]} -
            ↪  Accuracy: {acc_per_fold[i]}%')
208             if i == (len(acc_per_fold) - 1):
209                 print(
210                     f'Values:\n'
211                     f'Precision: {pre_per_fold[i]}\n'
212                     f'Recall: {rec_per_fold[i]}\n'
213                     f'F1 score: {f1_per_fold[i]}\n'
214                     f'Categorical accuracy: {cat_bin_acc_fold[i]}\n'
215                 )
216                 values = pd.DataFrame(data_dict)
217                 values.to_csv(f"{model_name}_values.csv")
218
219
        ↪  print('------------------------------------------------------------------------')
220         print('Average scores for all folds:')
221         print(f'> Accuracy: {np.mean(acc_per_fold)} (+-
        ↪  {np.std(acc_per_fold)})')
222         print(f'> Loss: {np.mean(loss_per_fold)}\n'
223               f'> Precision: {np.mean(pre_per_fold)}\n'
224               f'> Recall: {np.mean(rec_per_fold)}\n'
225               f'> F1 Score: {np.mean(f1_per_fold)}')
226
        ↪  print('------------------------------------------------------------------------')
227
228         # Use the validation data to generate prediction
229         # Use that prediction to calculate the model performance
        ↪  metrics
230         if model_name == 'stars':
231             y_pred = model.predict(X_val)
232             rec_score = recall_score(y_pred=np.argmax(y_pred,
            ↪  axis=1), y_true=np.argmax(y_val, axis=1),
233                                      average='weighted')
234             pre_score = precision_score(y_pred=np.argmax(y_pred,
            ↪  axis=1), y_true=np.argmax(y_val, axis=1),
```

```
                                    average='weighted')
        f1_measure = f1_score(y_pred=np.argmax(y_pred, axis=1),
        ↪  y_true=np.argmax(y_val, axis=1),
                              average='weighted')

        print(f'the recall score for the {model_name}')
        print(f'recall: {rec_score}')
        print(f'precision: {pre_score}')
        print(f'F1 score: {f1_measure}')

        return [rec_score, pre_score, f1_measure]

    else:
        y_pred = model.predict(X_val)
        y_pred = (y_pred > 0.5)
        rec_score = recall_score(y_pred=y_pred, y_true=y_val,
                                 average='binary')
        pre_score = precision_score(y_pred=y_pred, y_true=y_val,
                                    average='binary')
        f1_measure = f1_score(y_pred=y_pred, y_true=y_val,
                              average='binary')

        print(f'the recall score for the {model_name}')
        print(f'recall: {rec_score}')
        print(f'precision: {pre_score}')
        print(f'F1 score: {f1_measure}')

        return [rec_score, pre_score, f1_measure]


if __name__ == "__main__":
    main()
```

C Data exploration, plots and miscellaneous

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load the cleaned dataset, and the original data
df_clean = pd.read_csv('plumber_clean.csv')
m_cols = ['Plumber_ID', 'Stars', 'Title', 'Review', 'Date',
↪  'Reviewer_ID']
df_original = pd.read_csv('loodgieter_reviews.csv', sep=',',
↪  names=m_cols, skiprows=1, encoding='ISO-8859-1')
df_clean.head()

# Values of interest for Data Exploration
print(f'the size of the original dataset is: {len(df_original)}')
print(f'the size of the clean dataset is: {len(df_clean)}')
column = df_original['Review'].isnull().sum()
print(f'the number of ratings with missing stars is: {column}')
print(f'percentage missing is: {column / len(df_original) *
↪  100}%')
print(f'the skwewness is: {df_clean.Stars.skew()}')
print(df_original.Stars.describe())

# Load the datasets of the cross-validation steps
df_senti = pd.read_csv("sentiments_values.csv")
df_senti['F1 Score'] = round(
    2 * ((df_senti['Precision'] * df_senti['Recall']) /
    ↪  (df_senti['Precision'] + df_senti['Recall'])), 3)
df_senti['Folds'] = df_senti['Unnamed: 0'] + 1
df_senti.head(10)

df_stars = pd.read_csv("stars_values.csv")
df_stars['Folds'] = df_stars['Unnamed: 0'] + 1
df_stars.head(10)

# Mean of the cross-validation folds
print(np.mean(df_stars))
print(np.mean(df_senti))

# Plots of the accuracy and loss of the training
# Variables
acc_stars = df_stars['Accuracy']
acc_senti = df_senti['Accuracy']
los_stars = df_stars['Loss']
los_senti = df_senti['Loss']
```

```
folds = df_senti['Folds']

acc = plt.figure(figsize=(6, 4))
plt.plot()
# Accuracy subplot
plt.suptitle('Model Accuracy')
plt.plot(folds, acc_stars, 'k', label='Muliclass')
plt.plot(folds, acc_senti, 'k-.', label='Binary')
plt.legend()
plt.xlabel('Fold')
plt.xticks(folds)
plt.ylabel('Accuracy')
acc.savefig('fold_accuracy.png')
# Loss subplot

loss = plt.figure(figsize=(6, 4))
plt.plot()
plt.suptitle('Model Loss ')
plt.plot(folds, los_senti, 'k', label='Multiclass')
plt.plot(folds, los_stars, 'k-.', label='Binary')
plt.legend()
plt.xlabel('Fold')
plt.xticks(folds)
plt.ylabel('Loss')
loss.savefig('fold_loss.png')

acc.show()

# Figure that contains all 3 histograms
fig, (star_half, star, sent) = plt.subplots(3, 1, figsize=(10,
↪  10), sharex=True)

fig.suptitle('Distribution of the rating values')
star.set_title('Star values after cleaning')
star.hist(df_clean.Stars, color='k', density=False, bins=10,
↪  edgecolor='white')

sent.set_title('Sentiment values')
sent.hist(df_clean.Sentiments_Binary, color='k', density=False,
↪  bins=5, edgecolor='white')

star_half.set_title('Star values before cleaning')
star_half.hist(df_original.Stars, color='k', density=False,
↪  bins=10, edgecolor='white')

fig.show()
```

```
83
84  # Create histogram of data: Change values to fit the needed value
85  fig = plt.figure()
86  plt.plot(figsize=(10, 10))
87  plt.suptitle('Sentiment values')
88  plt.hist(df_clean.Sentiments_Binary, color='k', density=False,
    ↪  bins=2, edgecolor='white')
89  plt.xticks([0, 1])
90  plt.show()
91  fig.savefig('hist_sentiment.png')
92
93  # Sentiment
94  # Average scores for all folds:
95  # > Accuracy: 0.8703999999999998 (+- 0.11123147036697843)
96  # > Loss: 0.6660999999999999
97
98  # Star Rating
99  # Average scores for all folds:
100 # > Accuracy: 0.8846 (+- 0.12261745389625411)
101 # > Loss: 0.5745999999999999
```