TRIPADVISOR

Deep Learning with sequential data: predict hotel ratings based on reviews

TripAdvisor Team 2

Notebooks

1. Baseline model
2. Train embedding
3. Pretrained embedding: non-ordinal classification
4. Pretrained embedding: ordinal regression

Each of these notebooks includes:

* Preprocessing specific to the model
* Model (define, compile, fit)
* Confusion matrix
* Misclassifications

Report

1. Preprocessing
2. Baseline model: bag of words
3. Trained embedding (word2vec)
   1. LSTM
4. Pretrained embedding (Wikipedia)
   1. LSTM
      1. non-ordinal classification
      2. ordinal regression
5. Preprocessing

* The decision is made to only work with English reviews. So by using **language detection**, we removed the non-English reviews from both the content and rating list for training the model.
* **Preprocess** the data in Python, such that unnecessary noise could be eliminated as much as possible:
  + Remove the reviews with no rating
  + Convert text to lowercase
  + Remove punctuation
  + Remove stopwords
  + Remove numbers: We removed this because numbers can have multiple interpretations, it can have both a negative and positive connotation, which is in contrast to words like “good” that have a clear positive connotation.
  + Lemitization: Lemitization instead of stemming, because stemming often led to non-existing words, whereas, lemma is an actual language word.
* **Grouped split**, thus the same hotels can only occur in either the train, validate or test set. Furthermore, we make sure that they are shuffled. By this we mean that not all the reviews of the same hotel are direct successors and thus to increase the randomness of the data.
* We made a list with all the ratings. We use the keras function **‘to\_categorical’**: it transforms integers to a binary class matrix. In order to use this, we first need to transform the ratings (1, 2, 3, 4, 5) to (0, 1, 2, 3, 4). We do this, because the ‘to\_categorical’ functions starts from 0. If we wouldn’t do this, there would be a category too much. This is only necessary if the problem is treated as a classification problem.

1. Baseline model: Dense network with bag of words

As a baseline model, we will try to predict the overall rating for the different hotels based on the written text by using a bag of words.

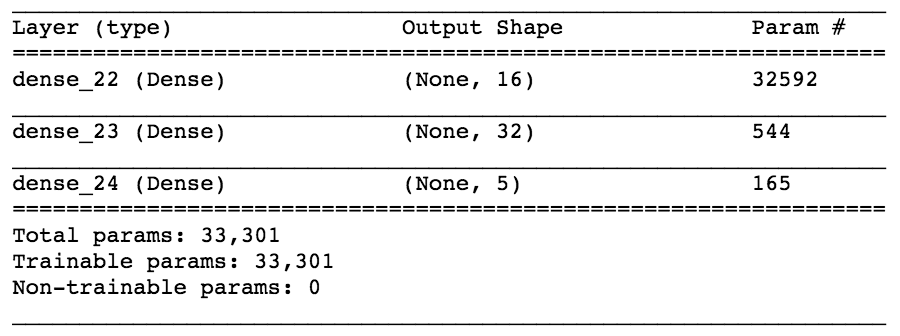
Bag of words

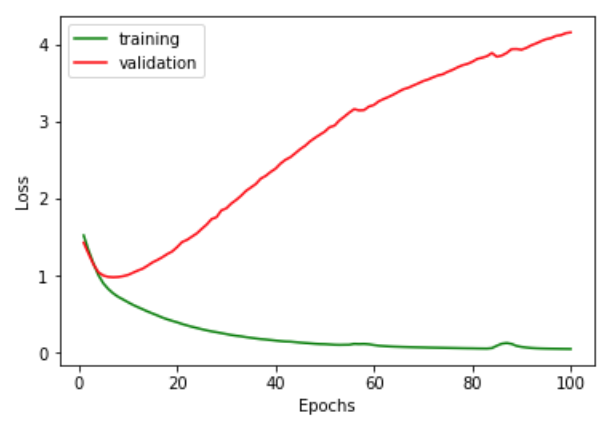
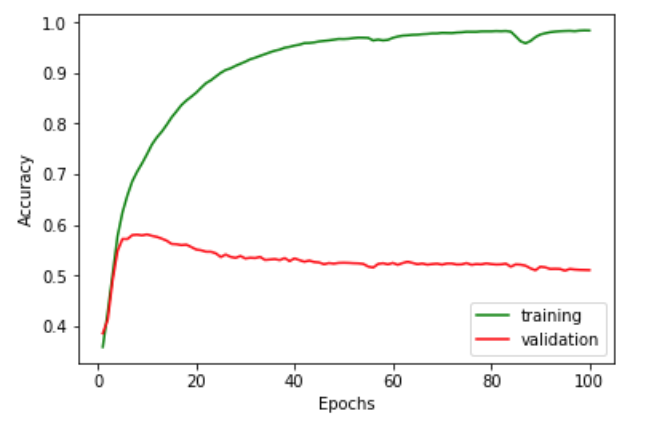
1. **Method** [1, 2]:
   * Make a vocabulary of all the words that occur in the training data.
   * Cut off the words that occur few times, otherwise we would be overfitting on them (min occurrence = 100).
   * Make a vector of this vocabulary which consists of for example 1000 words.
   * Remove the words from the reviews that are not in the vocabulary.
   * Convert reviews (sequences of words) to sequences of integers, where each integer stands for a specific word in a dictionary.
   * Similar to one-hot encoding: encode the reviews into vectors of 0s and 1s. e.g. if the review consists of 100 integers (so 100 words) you will have a vector with length 1000 (= length vocabulary) of almost all 0s and sometimes a 1, 2, 3... for the specific words of the vocabulary that occur in that review.
2. **Interpretation**[1, 2]:
   * Working with a bag of words means that you take all the words together, so forget about the meaning and the relations. You’re dealing with tokens that are interpreted as a set, not a sequence. The structure of sentences is lost. This is suffice as a baseline model.

Initial model

Now that the input data and targets are prepared, they can be feeded into a neural network.

|  |  |
| --- | --- |
| Number of neurons | Dense network with: 16, 32, 5 |
| Min occurrence | 100 |
| Number of epochs | 200 |
| Dropout | / |
| L1 / L2 / Maxnorm | / |
| Batch Size | 512 |
| Validation loss | 1.01163937697 |
| Validation accuracy | 0.58106964851 |





Hyperparameter tuning

We use 10% of the dataset for tuning. In order to tune our models and find the best hyperparameters, we work with a validation set. Since it is a classification task, we make decisions based on the validation accuracy. On top of that, we look at the loss graphs on which we can compare the validation loss with the training loss and thus, can conclude whether a model is overfitting or not. After tuning, the predictions are made for the test set. Since it is only our baseline model, there is less preprocessing and there are less details added to the model.

1. **Making network more powerful**

|  |  |  |
| --- | --- | --- |
| **Number of layers** | Validation loss | Validation accuracy |
| 16, 32, 64, 5 | 1.06412719241 | 0.581132075593 |
| 16, 32, 64, 128, 5 | 0.972415677262 | 0.574455733033 |

⇒ Adding 1 extra layer and thus making the network wider improves the validation accuracy a little bit from 0.58106964851 to 0.581132075593. Adding 2 layers reduces the validation accuracy.

|  |  |  |
| --- | --- | --- |
| **Number of neurons** | Validation loss | Validation accuracy |
| 160, 320, 640, 5 | 0.993149246316 | 0.587808418066 |
| 16, 16, 16, 5 | 0.992149446007 | 0.593033381816 |
| 32, 32, 32, 5 | 1.00575477317 | 0.5796806968 |

By adapting the number of neurons, a validation accuracy of 0.593033381816 can be achieved.

**Minimal occurrence:**

When making a bag of words, there is a decision on how often a word should appear to be in added into the bag. The smaller the minimal occurrence, the bigger the bag of words, the higher the validation accuracy and the sparser the vectors. Neural networks do not like these sparse vectors. Thus, there’s a trade-off between computation time and sparse matrices and accuracy. minimal occurrence transition from 100 to 2 barely improves the performance, but has a huge impact on time and sparsity of matrices. A minimal occurrence of 100 is chosen.

|  |  |  |
| --- | --- | --- |
| **Number of epochs** | Validation loss | Validation accuracy |
| 500 | 0.992149437252 | 0.593033381816 |

Increasing the number of epochs does not improve the performance.

|  |  |
| --- | --- |
| **Accuracy and Loss Curves:**   * We see that the training accuracy curve goes to 1 so the model is quite powerful. * The curves are diverging, the training loss is less than the validation loss so the model is clearly overfitting! |  |

1. **Regularization**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dropout** | Validation loss | Validation accuracy | Accuracy curve | Loss curve |
| 0.01 | 1.01160694432 | 0.58693759085 |  |  |
| 0.1 | 0.96016176593 | 0.58838896959 |  |  |
| 0.5 | 1.03615859171 | 0.57184325121 |  |  |

⇒ Randomly setting to zero a number of output features of the layer during training reduces the validation accuracy. So dropout is not added to our model.

L2, L1, maxnorm:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **L2 Regularization** | Validation loss | Validation accuracy | Accuracy curve | Loss curve |
| 0.0001 | 1.00254991868 | 0.593033381816 |  |  |
| 0.001 | 1.04468214806 | 0.594194484864 |  |  |
| 0.01 | 1.17033240217 | 0.588098693863 |  |  |
| 0.1 | 1.27945153488 | 0.512336720022 |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **L1 Regularization** | Validation loss | Validation accuracy | Accuracy curve | Loss curve |
| 0.0001 | 1.05270282561 | 0.600000000121 |  |  |
| 0.001 | 1.12614315594 | 0.599419448563 |  |  |
| 0.01 | 1.2513741252 | 0.557329463059 |  |  |
| 0.1 | 12.9540633485 | 0.380551523952 |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Maxnorm** | Validation loss | Validation accuracy | Accuracy curve | Loss curve |
| 2 | 1.27259794315 | 0.544532789754 |  |  |
| 1 | 1.22795902173 | 0.557329463059 |  |  |

→ Setting maximal norms to the weights does not improve the validation accuracy.

→L1 (0.0001) regularization gives the best validation accuracy until now. It only reduced overfitting a little bit.

Final model

|  |  |
| --- | --- |
| * Predictions on the test set have an MAE of 0.64797 * Validation loss: 0.903640818511 * Validation accuracy: 0.64022185906   There’s a lot of overfitting, but this is quite normal since very sparse and large vectors are fed into the model. The network overfits on this. The longer the vectors and the sparser they are, the more there will be overfitting. |  |

**Confusion matrix**

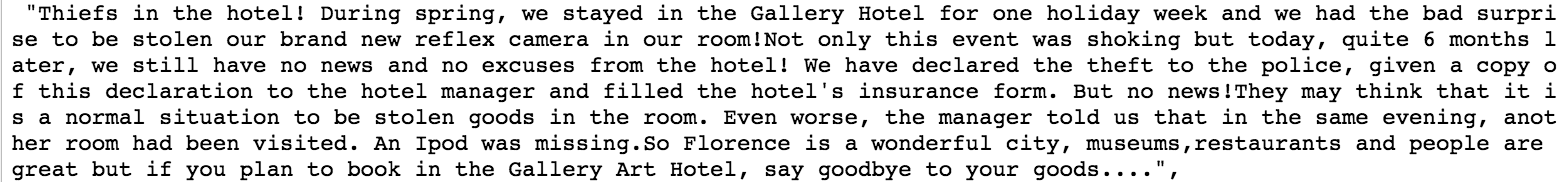
|  |  |
| --- | --- |
| First of all we want to mention that we have to increase each label in each of the confusion matrices by one in order to have its real rating score. For example label 2 means a rating score of 3.  The confusion matrix looks quite good, the diagonal line has the highest values (except for the true label of 2) which means that in most of the cases, the model will predict the true value correctly. Furthermore, if the model is wrong, it mostly mistakes by guessing one class higher or one class lower, as in the case of true label 2. This is a good sign. |  |

**Misclassifications**

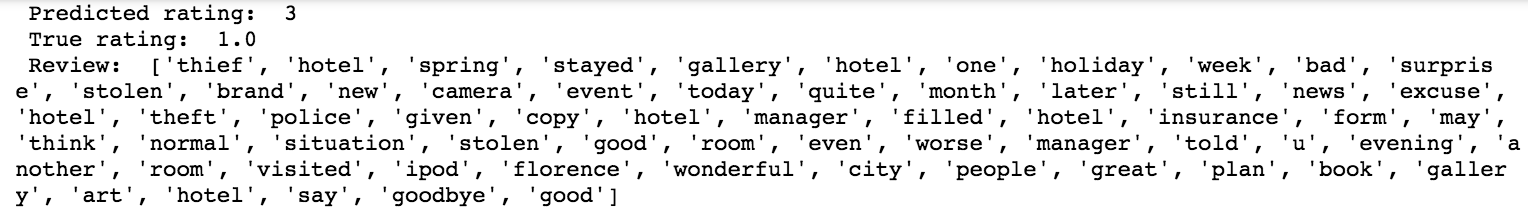
First 10 misclassifications:

* 4 misclassifications are acceptable: confuses ratings 4 and 5
* 6 misclassifications are not acceptable:
  + For example:

Predicted rating= 3, true rating = 1:

****

The model used the following words to predict the rating of this review:

****

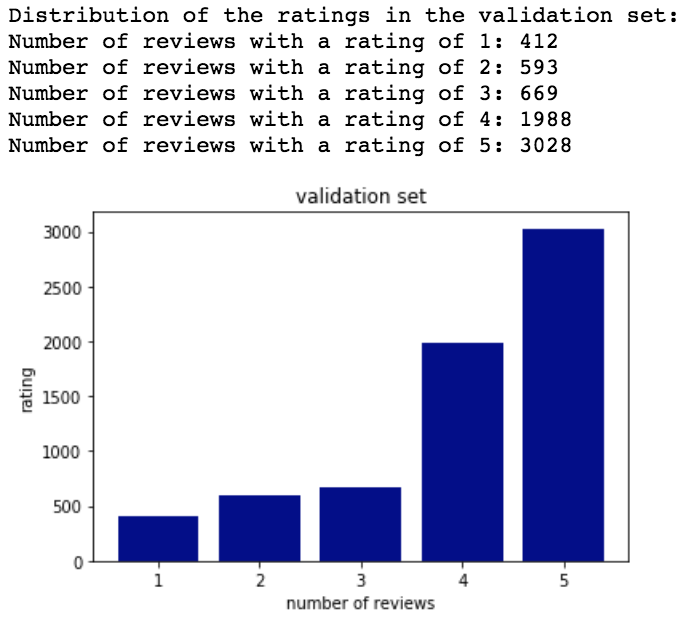
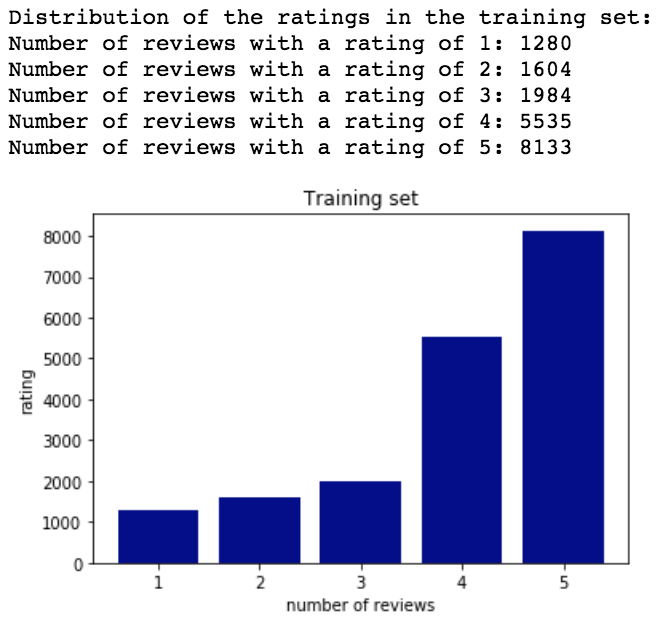
It is understandable that the model predicted a higher rating than 1 because a lot of positive words are in this review: e.g. good (2x), wonderful, great. And it is also normal that it did not predict a 5 because there are also some negative words: e.g. worse. Thus, it is understandable that the model predicts this review incorrectly. However, for a human being, this is a review that is clearly negative, there’s no hesitation in this. Our model is not good.

|  |  |
| --- | --- |
| As can be seen on this confusion matrix, it is clear that we are dealing with an imbalanced dataset (there are way more reviews with a higher rating in the dataset than reviews with a low rating). This is probably why the model is better trained in predicting these higher ratings. Since this was only the baseline model to get started and to gain some knowledge about the problem, we have not dealt with this before. However, since garbage in equals garbage out, it is necessary to deal with this imbalance. To do this, **weight balancing** is applied. Normally, each class carries equal weight, but now we want the minority classes to hold more weight. Thus, class weights are set to the imbalanced classes. Furthermore, besides telling the model to “pay more attention” to samples from an under-represented class, we have to check that the class distributions in the train and validation set are similar. |  |

1. Trained embedding (word2vec)

Instead of turning the data into sparse vectors, we pad the lists so that they all have the same length. Then they are turned into an integer tensor. An embedding layer is then chosen to be the first layer since it is capable of handling such integer tensors. This embedding is learned from the data and is more dense (in contrast to the sparse vectors of the bag of words model) [1]. Firstly, we train our own embedding using the Word2Vec algorithm [2].

When training the model, it is important that the distributions in the train and validation set are similar. Proof of this is shown below:



We check this for every model that follows.

1. LSTM

Referring to the problem of a bag-of-words of losing the structure of a sentence because it deals with a set of tokens instead of a list, we try an LSTM. Recurrent neural networks can learn representations for groups of words without being explicitly told about the existence of such groups. The network can learn what to store/throw away in the long-term state.

However, the LSTM takes extremely long to run when fitting the model (15 minutes per epoch when only training on 10% of the dataset). Thus, a cheaper/faster alternative is chosen: CuDNNLSTM. This is a fast LSTM implementation backed by cuDNN (<https://developer.nvidia.com/cudnn>).

Initial model

|  |  |
| --- | --- |
| Number of smart neurons | CuDNNLSTM with 100 smart neurons |
| Number of epochs | 100 |
| Dropout | / |
| L1 / L2 / Maxnorm | / |
| Batch Size | 512 |
| Validation loss | 1.3558 |
| Validation accuracy | 0.43405 |

Hyperparameter tuning

1. **Making model more powerful**

|  |  |  |
| --- | --- | --- |
| **Number of smart neurons** | Validation loss | Validation accuracy |
| 100 | 1.3558 | 0.43405 |
| 75 | 1.3645 | 0.41504 |
| 50 | 1.3557 | 0.42820 |
| 120 | 1.3603 | 0.42742 |

|  |  |
| --- | --- |
| Changing the number of neurons from 100 neurons does not increase the validation accuracy. Training accuracy goes to 1 but to make sure that our model is powerful enough we will make our curves smoother. To make the curves smoother, the following two techniques can be applied:    1. Increase batch size    2. Adapt learning rate  a. Too low: makes slower and risk of being stuck in a saddle point.  b. Too high: risk of jumping across minimum.  → make smaller but not too small! |  |

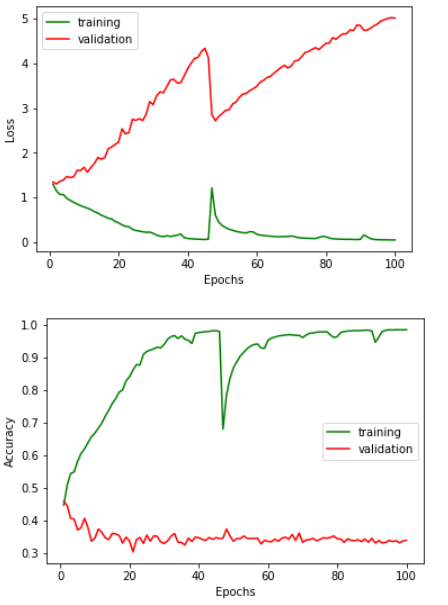
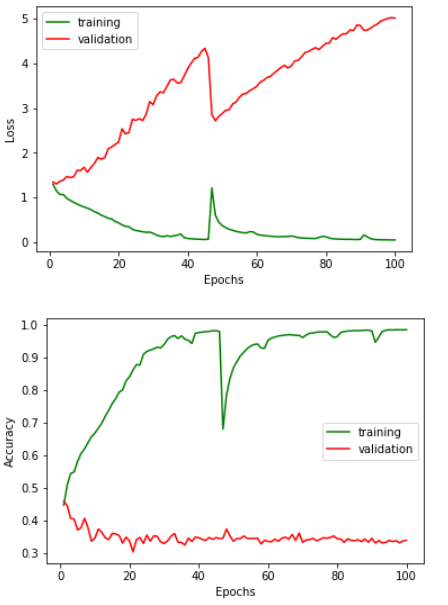
*Increase batch size:*

* from 512 to 1024:

Batch size is already high, by increasing this from 512 our kernel dies so we cannot   
 increase it.

*Adapt learning rate:*

|  |  |  |
| --- | --- | --- |
| **Learning rate** | Validation loss | Validation accuracy |
| 0.01 | 1.2851 | 0.44385 |
| 0.001 | 1.2970 | 0.45476 |
| 0.0001 | 1.2520 | 0.45107 |



→ curves are not really smoother but validation accuracy is better.

1. **Regularization**

As we can see, the curves are far away from each other (validation loss > training loss), this means that the model is overfitting. Now that we have tried to make the model more powerful, it is time to reduce this overfitting by doing regularization.

|  |  |  |
| --- | --- | --- |
| **Dropout** | Validation loss | Validation accuracy |
| 0.0001 | 1.3245 | 0.44531 |
| 0.01 | 1.2606 | 0.44953 |
| 0.1 | 1.3703 | 0.45010 |
| 0.5 | 1.2807 | 0.45324 |

⇒ Dropout of 0.5 improves the validation accuracy up to 0.45324.

|  |  |  |
| --- | --- | --- |
| **L1 regularization** | Validation loss | Validation accuracy |
| 0.001 | 1.3041 | 0.46341 |
| 0.01 | 1.4105 | 0.46356 |
| 0.1 | 1.6737 | 0.4441 |

⇒ L1 regularization (0.01) improves the validation accuracy up to 0.46356.

|  |  |  |
| --- | --- | --- |
| **L2 regularization** | Validation loss | Validation accuracy |
| 0.001 | 1.3367 | 0.45937 |
| 0.01 | 1.3505 | 0.46596 |
| 0.1 | 1.7670 | 0.4441 |
| 0.0001 | 1.7709 | 0.4441 |

⇒ L2 of 0.01 improves the validation accuracy up to 0.46596.

|  |  |  |
| --- | --- | --- |
| **Maxnorm** | Validation loss | Validation accuracy |
| 2 | 1.7709 | 0.4441 |
| 3 |  | 0.44411 |

⇒ Setting maximal norms does not improve the validation accuracy.

Final model with dropout of 0.50

|  |  |
| --- | --- |
| * Predictions on the test set have an MAE of 0.98811 * Validation loss: 1.7740 * Validation accuracy: 0.52155   The curves are extremely bumpy, look like bad models. Probably too much dropout. |  |

**Confusion matrix**

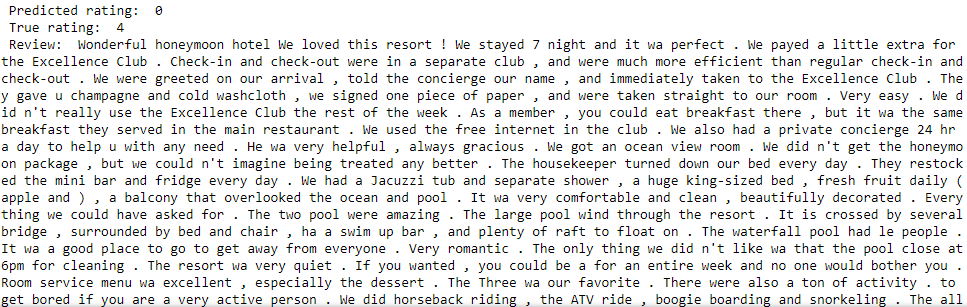
|  |  |
| --- | --- |
| From following confusion matrix we can derive that the model is best at predicting the label 4. Although there are still a lot of reviews labeled as 4 while its true label is 3. The model is less accurate in correctly predicting label 1. Label one is often mispredicted as label zero and as label four. The model is very bad at predicting label 2. The model is quite good in predicting the ‘extremes’. A sign of a ‘good’ confusion matrix would be a dark diagonal on the spots where the true label equals the predicted label. Which is not the case for this matrix. |  |

**Misclassifications**

First 10 misclassifications:

* 7 misclassifications are acceptable: confuses ratings 4 and 5
* 3 misclassifications not acceptable:
  + For example:

Predicted rating= 1, true rating = 5:

****

For a human being, this is clearly a positive review. It is not understandable that the model made such a big mistake, since there are not that many negative words used. This is again a sign of a bad model.

Final model without dropout

Predictions on the test have an MAE of 0.96277.

Validation accuracy dropped to 0.433766358048, which is actually even worse than the model before. However, the MAE on the test data is better.

**Confusion matrix**

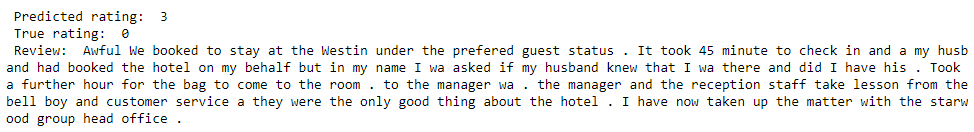
|  |  |
| --- | --- |
| This confusion matrix is even worse than the one of the model including dropout. For the true zero labels it predicted more of these labels as a 4 than it predicted zeros correctly.  Our model is best at correctly predicting 3’s and 4’s but it confuses the 3’s and 4’s a lot with each other. This confusion matrix makes clear that this model is not good. |  |

**Misclassifications**

First 10 misclassifications:

* 6 misclassifications are acceptable: confuses ratings 4 and 5
* 4 misclassifications are not acceptable:
  + For example:

Predicted rating= 5 and true rating = 2:



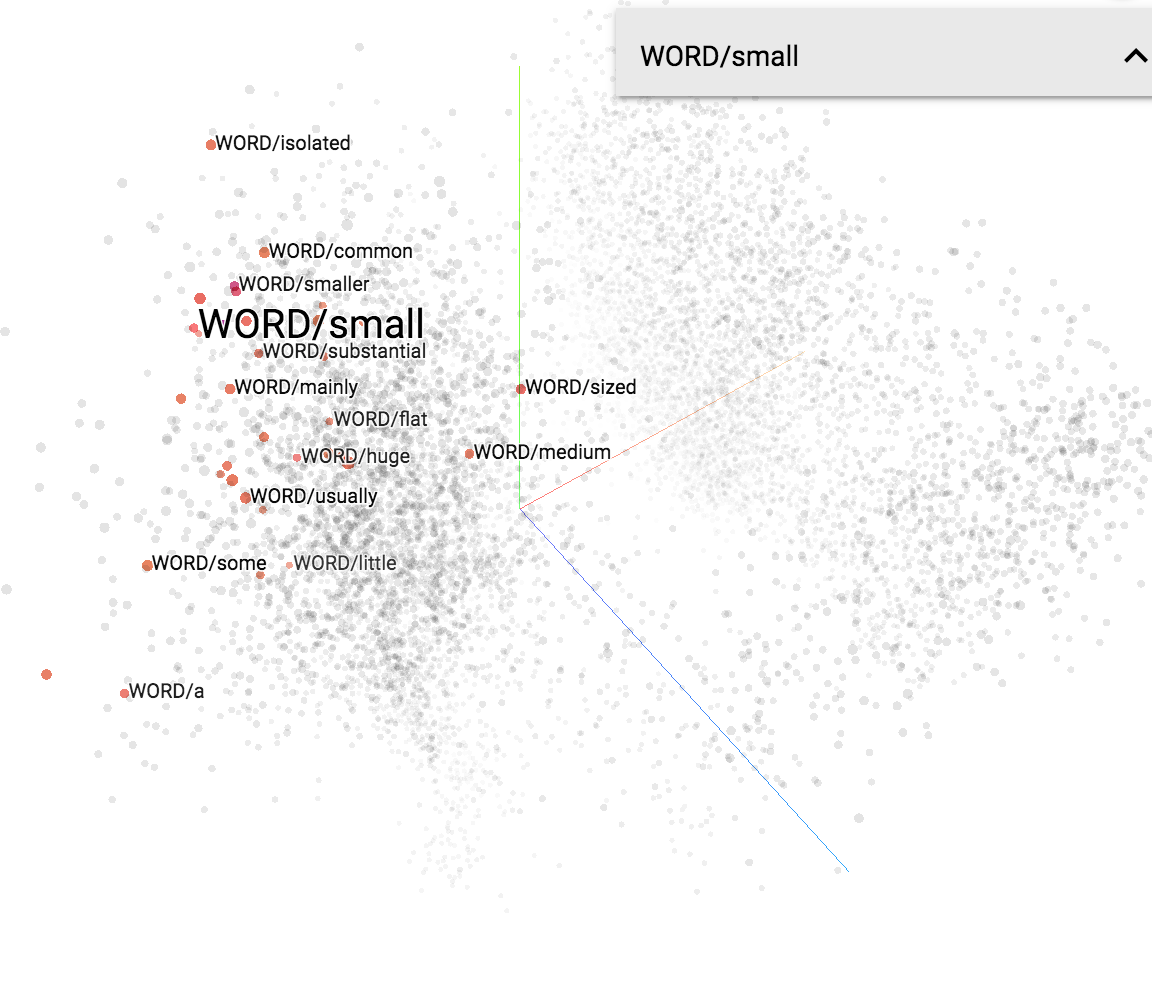
The fact that the model makes such a mistake here, is a sign of a bad model. For a human being, it is obvious that his is a negative review. Maybe the model got confused by the word ‘good’.

**Conclusion**

It can be concluded that training an embedding is not a good idea, this is probably not powerful enough to capture all the relationships between words like a pretrained embedding.

1. Pretrained embedding

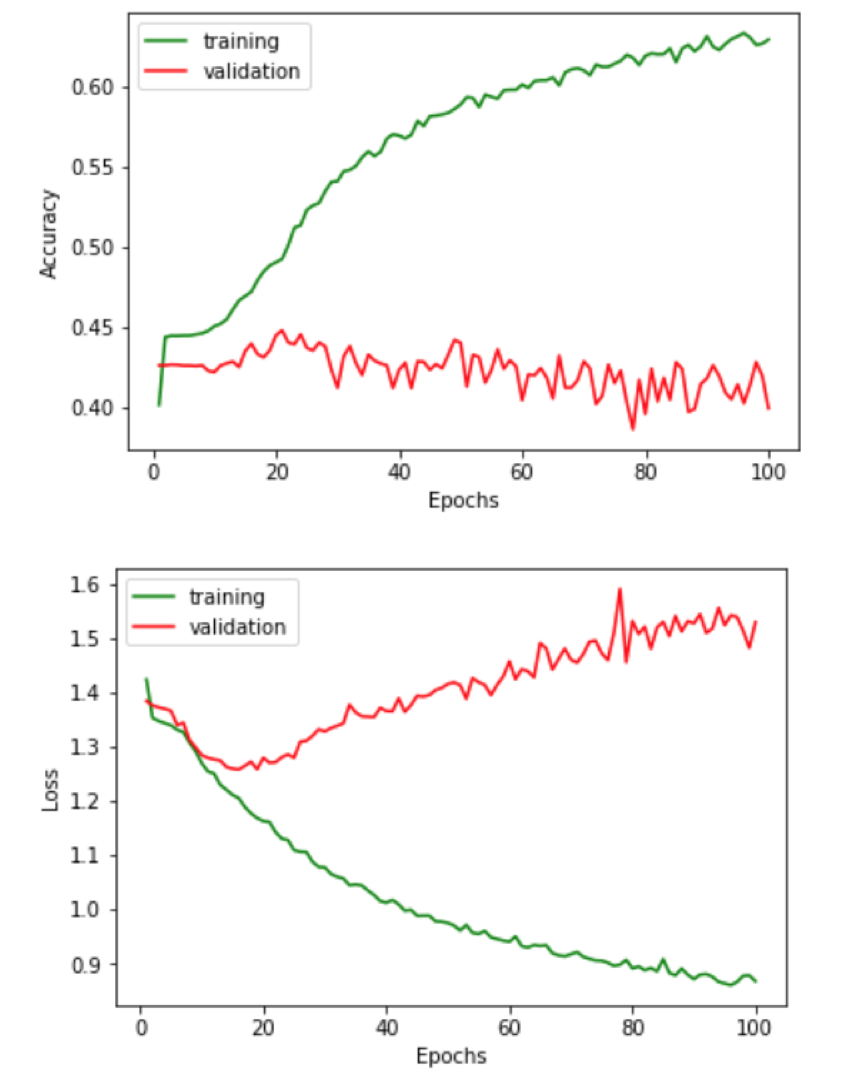
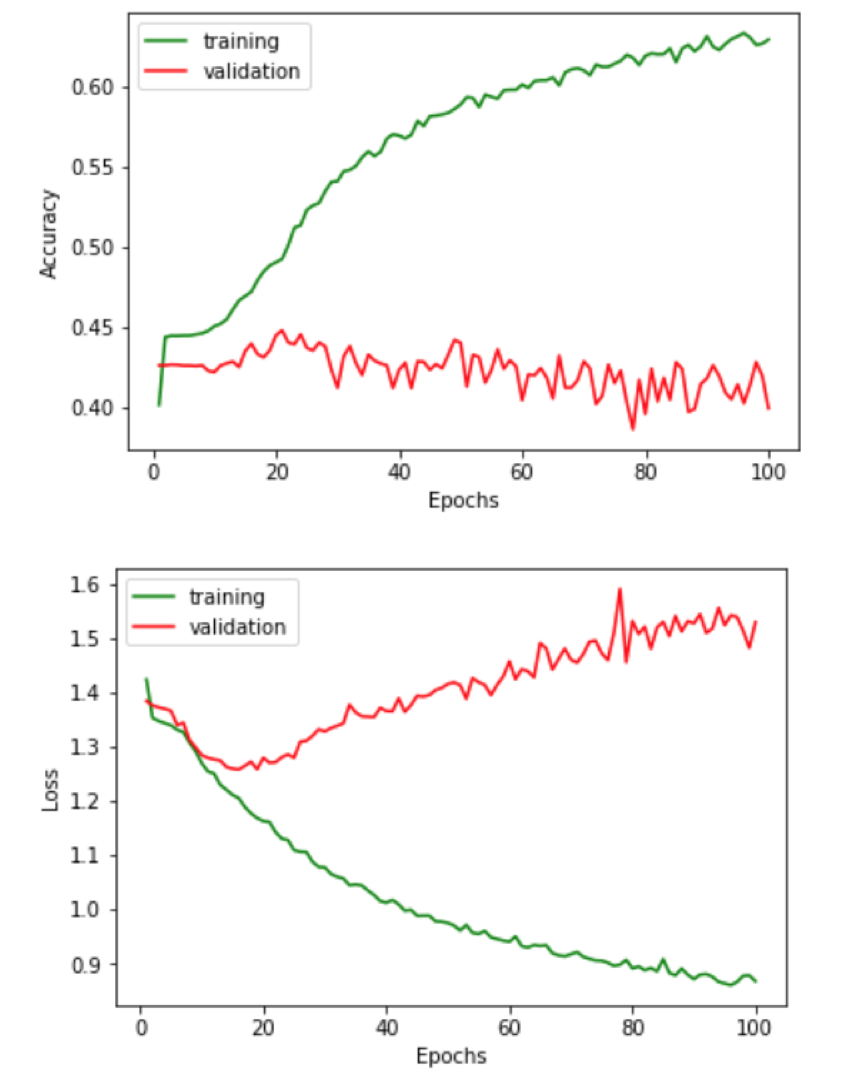
The pretrained embedding comes from: h[ttps://wikipedia2vec.github.io/wikipedia2vec/pretrained/](https://wikipedia2vec.github.io/wikipedia2vec/pretrained/) and is compatible with the format of Word2Vec. The name of the embedding is ‘enwiki\_20180420\_win10 (window=10, iteration=10, negative=15)’ and it has 100 dimensions. A graphical representation of the embedding:



LSTM as non-ordinal classification problem

Initial model

|  |  |
| --- | --- |
| Number of smart neurons | CuDNNLSTM with 20 smart neurons |
| Number of epochs | 100 |
| Dropout | / |
| L1 / L2 / Maxnorm | / |
| Batch Size | 128 |
| Validation loss | 1.2698500091 |
| Validation accuracy | 0.448130841121 |



Hyperparameter tuning

1. **Making model more powerful**

|  |  |  |
| --- | --- | --- |
| **Number of smart neurons** | Validation loss | Validation accuracy |
| 20 | 1.2698500091 | 0.448130841121 |
| 50 | 1.3705 | 0.46075 |
| 100 | 1.3231083168 | 0.465576324025 |

⇒ Increasing the number of smart neurons to 100 improves the validation accuracy to 0.465576324025. Making the number of neurons bigger, causes exhausted resources on the server. Thus a decision of 100 neurons is made.

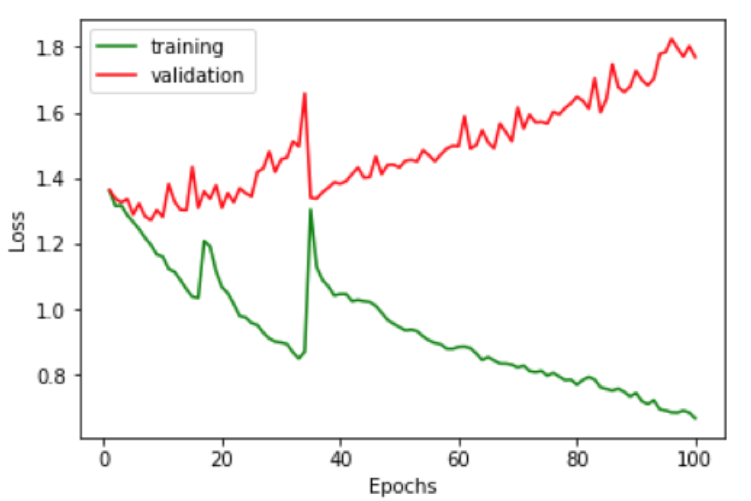
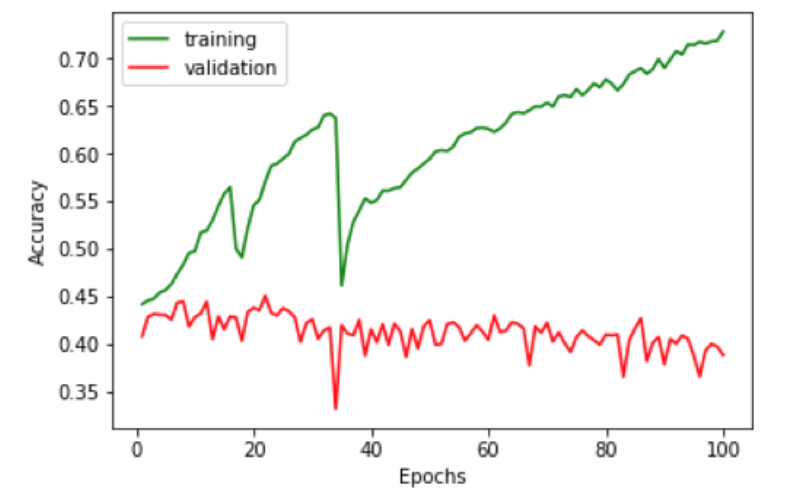
|  |  |
| --- | --- |
| Changing the number of neurons from 20 to 100, leads to a training accuracy that goes to one, so much more powerful. However, since the curves are quite bumpy, it is difficult to make conclusions about whether the model is powerful enough. To make the curves smoother, 2 techniques can be applied, as mentioned before:    1. Increase batch size    2. Adapt learning rate  a. Too low: makes slower and risk of being stuck in a saddle point.  b. Too high: risk of jumping across minimum.  → make smaller but not too small! |  |

*1. Increase batch size to 300*

- Validation accuracy of 0.450934579421, which is lower than before.

- Curves are not smoother, in fact, they have become even bumpier.

⇒ Decrease batch size back to 128.



*2. Adapt learning rate*

|  |  |  |
| --- | --- | --- |
| **Learning rate** | Validation loss | Validation accuracy |
| 0.01 | 1.3259 | 0.45748 |
| 0.001 | 1.2915 | 0.46557 |
| 0.0001 | 1.35186582562 | 0.454517133938 |

⇒ Curves do not get smoother and validation accuracy gets worse.

1. **Regularization**

As we can see, the curves are far away from each other (validation loss > training loss), this means that the model is overfitting. Now that we tried to make the model more powerful, it’s time for trying to reduce this overfitting with regularization techniques:

|  |  |  |
| --- | --- | --- |
| **Dropout** | Validation loss | Validation accuracy |
| 0.0001 | 1.3967 | 0.45248 |
| 0.01 | 1.4819 | 0.45187 |
| 0.1 | 1.3000 | 0.45654 |
| 0.5 | 1.46601392606 | 0.453271028 |

⇒ Randomly setting to zero a number of output features of the layer during training reduces the validation accuracy. So dropout is not added to our model.

|  |  |  |
| --- | --- | --- |
| **L1 regularization** | Validation loss | Validation accuracy |
| 0.001 | 1.3407 | 0.43754 |
| 0.01 | 1.4239 | 0.42852 |
| 0.1 | 1.8131 | 0.42741 |

|  |  |  |
| --- | --- | --- |
| **L2 regularization** | Validation loss | Validation accuracy |
| 0.001 | 1.3566 | 0.43502 |
| 0.01 | 1.4068 | 0.44704 |
| 0.1 | 1.3769 | 0.42741 |

|  |  |  |
| --- | --- | --- |
| **Maxnorm** | Validation loss | Validation accuracy |
| 2 | 1.3631 | 0.45576 |
| 3 | 1.3922 | 0.45453 |

⇒ None of the regularization techniques (dropout, penalizing, setting maxnorm) improves the validation accuracy.

Final model (all data)

|  |  |
| --- | --- |
| * Predictions on the test set have an MAE of 1.04388 * Validation loss: 1.16228640844 * Validation accuracy: 0.510920121335   The validation loss curve is way higher than the training loss curve, which means that the model is heavily overfitting. Also the MAE on the test set is quite high. |  |

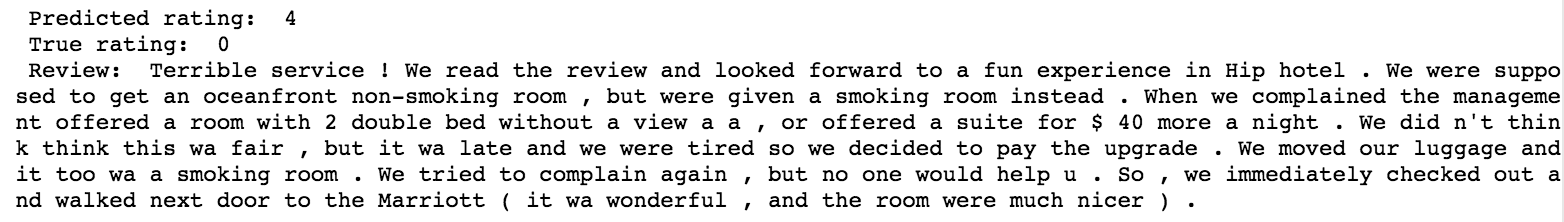
**Confusion matrix**

|  |  |
| --- | --- |
| The confusion matrix shows that the model is good at predicting the two extreme classes. Thus it will classify very negative reviews as negative and extremely positive reviews as positive. This is a good thing. However, the model is bad at predicting the ratings in between. It is still doing quite okay for ‘3’ but the model is entirely wrong when it has to predict ‘2’. For ‘1’ it is acting strangely, or it predicts that the review is extremely bad, or that it is extremely good. |  |

**Misclassifications**

First 10 misclassifications:

* 4 misclassifications are acceptable: confuses ratings 4-5, 3-4, and 1-2
* 6 misclassifications not acceptable:
  + For example: predicted rating = 5, true rating = 1



This review consists of both positive and negative words.

* positive words: fun, hip, upgrade, wonderful, nicer
* negative words:terrible, complained, tired

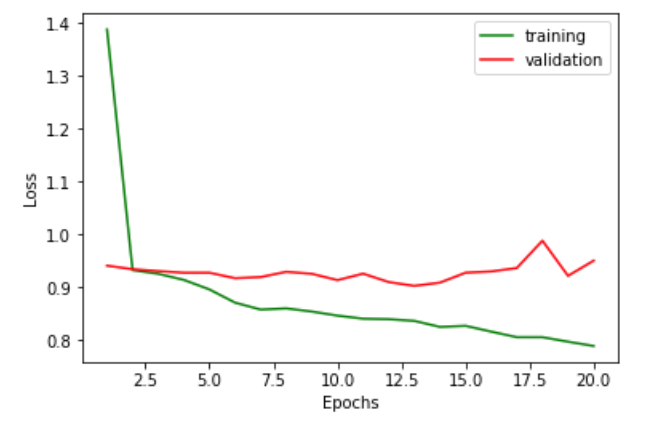
In our opinion the huge mistake of predicting 5 when the true rating is 1, is due to 2 reasons. There are more positive words in this review than negative and the positive words’ positive connotation is way stronger than the negative connotation of the negative words. We also think that this review is really hard to predict, since it’s a review in which a customer is both positive and negative, if the sentence between brackets wouldn’t have been there, then there’s a big chance that the model would have predicted a negative rating. For a human being it is clear that this is a negative review about this hotel. But it is understandable that the model cannot guess this, since it’s actually a review for 2 hotels (the bad hotel and then the good hotel Marriott).

LSTM as an ordinal regression problem

The problem of this project could also be considered as an ordinal regression task. Until now, the model is tuned in the interpretation of being a non-ordinal classification task, thus working with non-ordered categorical ratings. Now the model will be tuned based on the validation mean absolute error with the interpretation of being an ordinal regression task. Thus from now on, a mistake of predicting 1 for the true rating of 5, will be larger than predicting 4 for this rating.

Initial model

|  |  |
| --- | --- |
| Number of smart neurons | CuDNNLSTM with 20 smart neurons |
| Number of epochs | 20 |
| Dropout | / |
| L1 / L2 / Maxnorm | / |
| Batch Size | 128 |
| Validation MAE | 0.902517946867 |



Hyperparameter tuning

Based on 10% of the data.

1. **Making the model more powerful**

|  |  |
| --- | --- |
| **Number of smart neurons** | Validation MAE |
| 10 | 0.925666288688 |
| 20 | 0.902517946867 |
| 35 | 0.915816000333 |
| 50 | 0.907691138524 |

⇒ Increasing the number of smart neurons does not improve the validation MAE.

Both increasing the number of epochs to 50 and adapting the batch size had no positive impact on the validation MAE.

1. **Regularization**

As the number of epochs increases, the validation loss curve gets higher compared to the training loss curve. This means that the model is overfitting, thus there’s a need for regularization techniques.

|  |  |  |
| --- | --- | --- |
| **Dropout** | Validation loss | Loss curves |
| 0.05 | 0.900417648829 |  |
| 0.1 | 0.896832175438 |  |
| 0.25 | 0.902492393897 |  |

⇒ Dropout of 0.1 improved the validation MAE to 0.896832175438 and brought the curves closer together, thus reduced overfitting.

|  |  |  |
| --- | --- | --- |
| **L1 Regularization** | Validation loss | Loss curves |
| 0.0001 | 0.886513187794 |  |
| 0.001 | 0.936503243905 |  |
| 0.01 | 0.937031043951 |  |

⇒ L1 (0.0001) regularization improved the validation MAE to 0.886513187794

|  |  |  |
| --- | --- | --- |
| **L2 Regularization** | Validation loss | Loss curves |
| 0.0001 | 0.894314076809 |  |
| 0.001 | 0.885523107877 |  |
| 0.01 | 0.916994185631 |  |

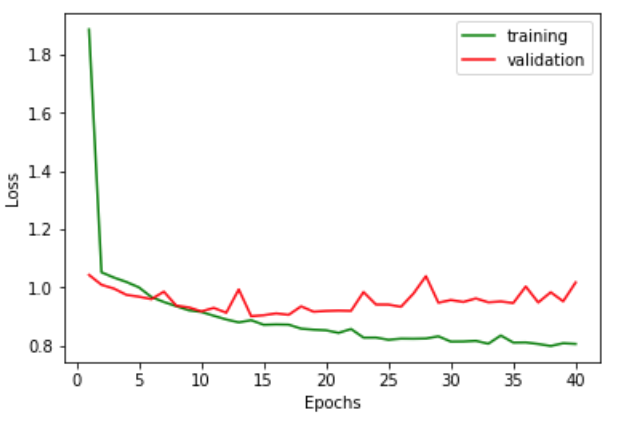
⇒ L2 (0.001) regularization improved the validation MAE even more to 0.885523107877. So L2 regularization (0.001) is added to the model.

|  |  |  |
| --- | --- | --- |
| **Maxnorm** | Validation loss | Loss curves |
| 2 | 0.897964049303 |  |
| 3 | 0.897964049303 |  |

⇒ Setting maximal norms does not improve the validation accuracy

1. **Make more powerful**

After doing some regularization on a limited number of epochs, we check whether increasing the number of epochs influences the validation MAE. Increasing the number of epochs to 40 did not improve the validation MAE (0.885523107877) but it is useful have a better view of what the curves are doing.

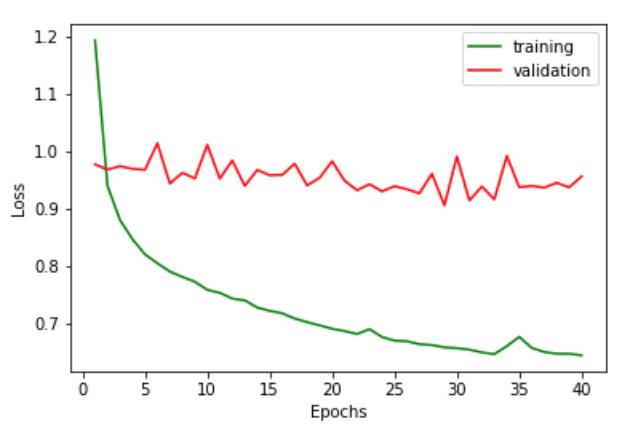


1. **Bumpy curves**

The validation loss curve is quite bumpy. As mentioned earlier this could be solved by making the batch size larger or making the learning rate smaller.

|  |  |  |
| --- | --- | --- |
| **Batch size** | Validation MAE | Loss curves |
| 256 | 0.882993929661 |  |
| 512 | 0.885831864522 |  |

⇒ Increasing the batch size makes the curves sufficiently smooth and a batch size of 256 also improved the MAE to 0.882993929661 .

Final model (on all data)

|  |  |
| --- | --- |
| * Predictions on the test set have an MAE of 4.00740 * Validation MAE: 0.875831034594   The validation loss curve is higher than the training loss curve, which means that the model is overfitting. Also the MAE on the test set is very high. |  |

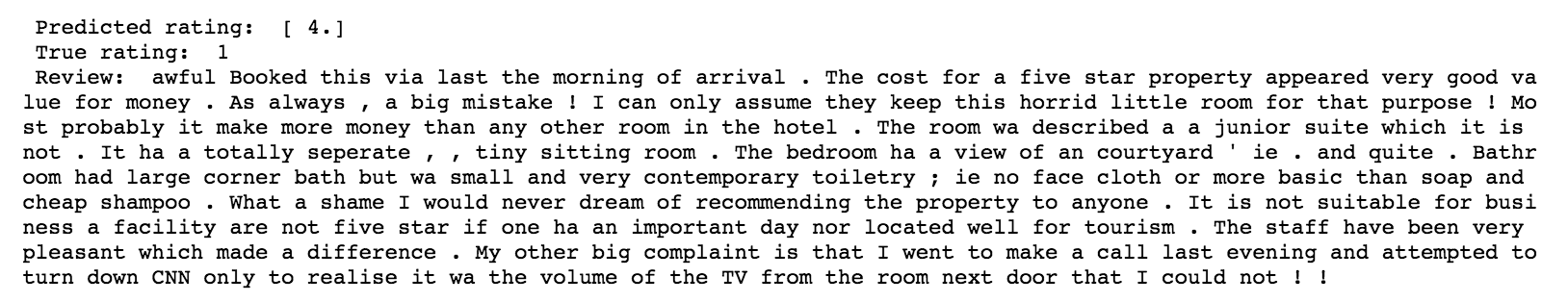
**Confusion matrix**

|  |  |
| --- | --- |
| The confusion matrix shows that the model is good at predicting the last 2 classes and also quite good at predicting the middle last class, since it mostly only mistakes with 1 class higher. However, it’s very bad at predicting the first 2 classes. |  |

**Misclassifications**

First 10 misclassifications:

* almost always predicts positively (rating 4 or 5)
* 4 misclassifications are acceptable: confuses ratings 4 and 5
* 6 misclassifications not acceptable:
  + For example:



It is possible that this model saw this as a positive review because of the words: good, star, dream, recommending, suitable, pleasant… However, the model should not give it such a positive rating since there are also negative words such as: awful, mistake, horrid, complaint… A predicted rating of 2-3 would be more acceptable in this case. In this model, the predicted ratings are rounded. It is also possible that rounding the values makes our MAE worse, maybe this model predicted a 3.6 (rounded to 4) for this review, this would have been a little bit more acceptable. Now that we gained this insight, we predict the test data, without rounding. Now Kaggle gives an MAE of 0.93586 which is much better!

**Conclusion**

We can conclude that this model has the preference to predict higher classes, in either case. Although there were class weights given to give the minority classes (low ratings, higher weights). This together with the very bad MAE (for rounded values) on the test set and the confusion matrix shows that in this case, tuning the model as an ordinal regression task, is not the best approach when rounding the variables. However, when not rounded, the MAE on the test set is better than the categorical variant of this model.

1. General conclusion

Now that all the models are studied in depth and compared to each other, we can say that our final model has the most opportunities in terms of improving the MAE. Thus, looking back to the models we have created, it can be concluded that ordinal regression could be the approach with the most potential in successfully predicting ratings based on sequential data: reviews. Focusing on ordinal regression instead of non-ordinal classification would be the advice we give to others with a similar project. We are aware of the fact that the MAE of the baseline model is better than the other models, but based on the misclassification we studied, it was clear that the baseline model made huge mistakes for reviews that were obviously positive or negative for human beings. The misclassifications of the models that followed made more sense and seemed like ‘smarter’ mistakes.

Things that are not included in this project, but are probably worth considering:

* Instead of removing the non-English reviews, they could be translated into English, such that we can develop a model that not only predicts English reviews. We could also use pretrained embeddings containing different languages instead of translating every word.
* Use of subratings as auxiliary targets such that the model trains multiple outputs. Here you would need to give weight to each output.

1. Literature

[1] Chollet, F. (n.d.). *Deep learning with Python*.

[2] Géron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow*. O’Reilly Media.