Project

Machine Learning Methods in Economics Course Winter Semester 2020/21

University of Hamburg

Prof. Melanie Krause, Ph.D.

NATURAL LANGUAGE PROCESSING WITH AMAZON PRODUCT REVIEWS

submitted by: Georg Zhelev

Abstract

This project seeks to determine for what pre-processing level, data encoding structures and feature-size a machine learning method can accurately predict the sentiment of amazon reviews. It demonstrates that a learned word embedding, using an artificial neural network, is a very efficient way to structure data that allows for scaling to a large data set. The project bordered the deep learning field due to encountered problems such as running out of memory and the necessity to structure data in a dense form. The question was tested using small and large sample sizes of amazon reviews and their star ratings. Python was used for the implementation, while computation was assisted by a remote server with the ability to execute parallel calculations.

Contents

1 Introduction

1.1 Question to be Answered

The goal of this project is to predict the sentiment of amazon reviews using test data, which holds out the rating. A second goal of this project is to engage a deep learning model on a large data set for text analysis. Further, the project seeks to test different scenarios to determine the impact of text pre-processing, varying data structures and feature sizes on the computational-time and accuracy. To do that, three deep learning models are implemented as well as a baseline model.

1.2 Data Description & Summary Statistics

The dataset consists of a total of 3,6 million Amazon customer reviews and their 1-5 star ratings, of which smaller samples are used in the project. The reviews serve as the input texts while the labels serve as the target. The data set is retrieved from Kaggle [1]. Kaggle's own notebook-framework is used, which simplifies the loading of the data set, because no downloading is necessary. Further, it allows using a remote server's computational power, such as a CPU and also a GPU to execute some of the calculations in parallel. This achieves fast computation on sample sizes in the hundredths of thousands.

The data set has already been prepared for text analysis. Reviews and labels are listed on one line, where reviews with a star rating of four to five stars are labeled with "__label__2", while reviews with one to two stars are labeled as "__label__1". Therefore there are no neutral ratings (3 stars) in the data set. This transforms the classification problem into a binary problem. The reviews also have titles which are included. The data set is already balanced, meaning there is an equal distribution between positive and negative reviews.

2 Theoretical Framework

2.1 Why Machine Learning

Choosing a machine learning method to analyze the data is an efficient way to quickly identify trends and patterns. Once set up, it works automatically, and can adapt to new data. Through their iterative aspect machine learning methods produce reliable and repeatable results [8]. Compared to a human being reading the reviews then assessing if positive or negative, a machine learning model takes advantage of the calculating power of a computer. As Mueller and Guido point out, it could very well be possible to create a system of *if-else* rules that label a review as positive or negative [7]. For example dependent on if words such s "great" or "bad" came up, but this system would not be adaptable to a different task.

2.2 Choice of Method & Evaluation Criterion

The machine learning method applied is an artificial neural network, which will be referred to as neural network in the rest of the paper. A neural network is particularly good for processing large amounts of data, due to its ability to execute parallel computations. It can further learn non-linear complex relationships [5]. In comparison to a logistic regression is a neural network not as easy to interpret. However it allows the implementation of an embedding layer that learns associations of words with other words.

A logistic regression is applied as a baseline model, because it is the first choice when it comes to classification. It is easy to interpret, because the prediction is based on a probability. A logistic regression may overfit when the amount of features is large [6]. That could be the case when training a logistic regression on few text observations with a large vocabulary. Therefore regularization should be used. A logistic regression does not allow for the learning of a word embedding. The evaluation criterion for all models is the accuracy score, which is made up of the sum of the correctly predicted true and false positives divided by all predictions.

3 Methodology

Several of the data loading and pre-processing steps presented in this paper were inspired by the discussion and resources on the internet page of the data set [4]. These resources were invaluable as a starting point on which the project is build upon.

3.1 Unpack, Load and Decode Data

First the data is loaded by accessing Kaggle's directory structure where the amazon data set is saved. Then the data is unpacked and each line is read until a specific number of bytes is reached. This is done, because not the entire data set will be used in the project, but a sub-sample instead. Lines are read in an ascending manner. The bytes are calculated by multiplying a constant with the number of lines one wants to extract from the whole data set. Each line represents one observation. Each line is then stored as an element of a list. Looking at the first line, it contains the label, the title and the text of a single review. Then the strings are decoded from Unicode characters to the standard UTF-8. In the same loop labels and texts are extracted. Viewing the prepared data, one can see that so many lines were read as specified. Converting the labels and texts to a data frame using pandas allows for easier viewing of the data. For example viewing the value counts shows that the sample is balanced.

3.2 Pre-Process

Pre-processing is important to prepare text data for a machine learning model. Text data has upper case letters, punctuation as well as connector words called stop words such as "the" "and" "to", which don't directly contribute to determining if a review is positive or negative. The goal is to remove these, while leaving important content. Pre-processing text may improve the performance of machine learning models. In this case pre-processing refers to lower casing, removing of punctuation and non-English characters and stemming. Stemming is reducing a word to its stem, while removing the conjuration. All words were normalized to lower case. Then using the regular expressions (REGEX) module non-word characters such as numbers and punctuation were replaced with white spaces. Characters that don't belong to the English language were also removed. This was the baseline pre-processing done. Further pre-processing using the Natural Language Toolkit (NLTK) was done to see if the performance of the model would increase. This includes removing stop words, stemming as well as implementing NLTK's own method for removing punctuation and non-English language characters. In order to remove stop words, the texts were tokenized by splitting words based on white space. Using NLTK's stop word-method, only words are kept which are not included in the stop words list. Stemming was also implemented.

3.3 Split, Tokenize and Encode Data

Data was split into training and valuation sets, made up of appropriate labels (taget) and texts (features). The train set was used to train the model, while the valuation set to determine how good it is trained. The pre-processed test set was held out until the models have been trained. Tokenization refers to the splitting of words into tokens and learning of a vocabulary based on their frequency. By tokenization a vocabulary is created that assigns a word index to each word in the corpus. The corpus is the entire text-data set. The user can set the maximum number of words in the vocabulary. Then the vocabulary is truncated after the most common words in the corpus. Common words remain and uncommon words are removed. The length of each review is reduced during tokenization, because only the x-most common words are chosen. If the words in a single review are unique and don't appear in other reviews, they will be dropped. The number of unique words is the total number of words found in the corpus, while the size of the vocabulary is set by the user. The user can choose to not drop any words and use all unique words in the corpus. The vocabulary is fit on the train data and then applied to the valuation and test data. This guarantees that the similar words in the three samples become the same index. The tokenized words are encoded in two ways: as integers or one-hot-encoded.

Integer Encoding

When applying the fitted dictionary to the train sample, each review in the corpus is transformed to a sequence of integers. Each word is represented by an integer. Also referred to as ordinal encoding, the order is based on word frequency in the corpus. One can access the word index (vocabulary) to see the integers assigned to words. The most common ones come first while the least common ones, which are often misspellings, come last. Each text is now represented by a numerical vector of integers. The length of the vector varies across reviews, because originally they are of different length. The vector is however shorter than the original tokenized text, because only the most common words were kept.

One-Hot-Encoding

A second possibility to transform text data into numerical data is by structuring it in a sparse matrix using binary (0-1) encoding. The rows of the matrix represent the observations (review-texts) and the columns represent the words. Each element is either a 1 when that particular word is included in the text or a 0, when the word is not part of the review. The matrix is sparse, because very few elements are non-zero. The number of columns depends on how many words the user decided to keep in the vocabulary. In summary the tokenized text data was encoded into numerical data in the two ways described above. A machine learning model can now work with this data.

Pad Sequence Data

In order to pass the integer encoded data into a fully connected neural network, it is necessary that each sequence of integer encoded words is of the same length. Because the reviews vary in length a technique called padding is used to normalize them to the same length. Shorter texts are padded with a character 0. This character is not assigned to any words in the word index therefore it will not affect the training. Zeros are added before texts that are too short. The longest review is used as a measuring stick to pad all other reviews to its size. Without padding the sequence data, the neural network that uses it will not work.

3.4 Train Models

Neural Network with a Learned Embedding Layer & Neural Network with Integer Encoding

The goal was to have a simple as possible architecture while maximizing the prediction accuracy and minimizing the computational time. The neural network is made up of an embedding layer, a pooling layer, one hidden layer with 10 nodes, and one node in the output layer for making binary classifications. The embedding layer attempts to map the meaning of language into geometric space. This is done by building a numeric vector for each word in the dictionary, such that the distance to other word vectors captures the linguistic relationship between the words. Words that are similar will appear close to each other in the embedding space. For example as seen in Figure 1, the distance between the words *man* & *woman* and the words *king* & *queen* is the same.

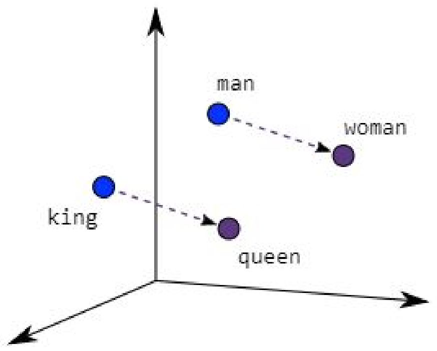

Figure 1: Embedding Word Example
source: developers.google.com [3]

The embedding layer maps the integer-representation of each word to an n-dimensional vector. The layer has weights that are trained like the weights of other layers in the neural network. The input to the embedding layer is the number of distinct words in the vocabulary. The output is the size of the embedding vector, which the user can freely choose. The length of each input sequence, which was normalized using padding is also entered. A trained word embedding is specific to the data set it was trained on. For completeness a neural network was also fitted on the integer encoded data without learning an embedding, in order to see if this encoding type could yield promising results.

Neural Network with One-Hot-Encoding & Logistic Regression as Baseline Model

A second neural network is fitted, which uses the one-hot-encoded data. The goal is to compare the performance and computational demands between the learned embedding and the one-hot-encoded data. Aside from the embedding layer missing, this neural network hast the same structure as the one described above. The input to this neural network is the one-hot-encoded matrix, which dimensions are the number of observations and the vocabulary size. Compared to the learned embedding, the one-hot-encoded matrix has a very long vector (size of the vocabulary) with a binary encoding for each word. In contrast the user can set a small size of the learned embedding vector, which is dense and is learned by the neural network. Figure 2 shows a graphical representation of the two types of encoding. Finally a logistic regression, which is also fed the one-hot-encoded data, is used as a baseline model.

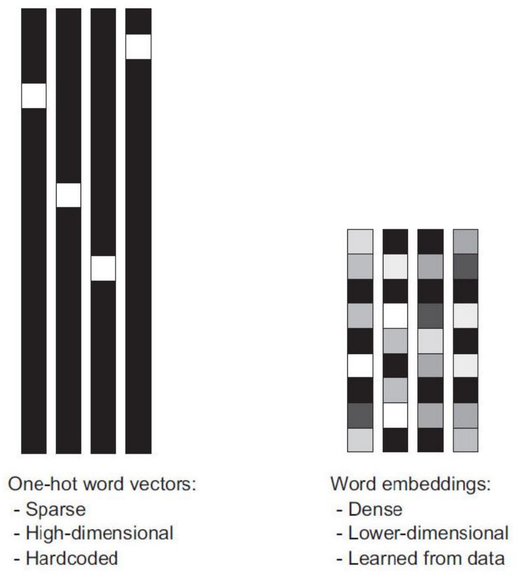

Figure 2: One-Hot-Encoding vs. Learned Embedding
source: Chollet, 2018: Deep Learning with Python [2]

Model Hyperparameters

The neural networks were run with 4 epochs, because it was found that after 4 the valuation accuracy plateaued. The mini-batch size was chosen to be 512 in order to speed up computational time. Lower batch sizes did not lead to better results. The optimizer *adam* (adaptive moment estimation) was used, which in contrast to stochastic gradient descent allows a variable learning rate for the updates of weights inside the neural network. The loss function was binary cross-entropy, which is the standard choice method for a binary classification problem. Specific to the neural network with the embedding layer, 100 was the chosen dimension to map words to. The logistic regression used an *L2* penalty with a parameter $C = 1$. Grid search could have been used to optimize the training of the models as well as running each model multiple times and looking at the mean accuracy. However the goal in this project was to focus on the encoding rather than achieving the best accuracy.

4 Results

The four models were run for different sample sizes, for different encoding types, for different sizes of the vocabulary and for different levels of pre-processing.

4.1 Sample Size and Data Encoding Study

The performances of the three encoding methods (one-hot, word embedding & integer) were analyzed. For the case of the integer encoding, the neural network could not distinguish between a positive or a negative review more than tossing a coin. Where one-hot-encoding is applied for sample sizes greater than 300 thousand and vocabulary size greater than 4000, the remote server ran out of memory (OOM). This means that a sparse matrix bigger than 300k times 4k can not be created. Therefore the neural network without an embedding layer and the logistic regression could not be applied to a larger data sample. As seen in Table 1 for sample sizes under 300k there was only a small performance difference between the three models. It seems that a learned embedding is about equally as good as a one-hot-encoding. The performance of the baseline model is only one percent lower than that of the two neural networks. In addition, the neural network with the learned embedding was tested on up to 2,5 million observations. The *Run-Time* represents the time to execute all three models. Where there is an OOM error the run-time represents only the fitting of the neural network with the learned embedding.

Table 1: Sample Size Study (Vocab. Size: 4000)

Size Train (k)	Size Test (k)	NN w/ Embed.	NN w/ One-Hot	LogReg	NN w/ Int.	Run-Time (Min.)
50	10	88%	88%	87%	0.51	0.7
100	100	88%	88%	88%	0.51	2
300	300	88%	88%	88%	0.51	5.3
2500	400	90%	OOM	OOM	N/A	11

*Using Kaggle remote CPU and GPU

An aspect which added to the memory issues was using different variable names for the train sample during different steps of the process. Doing that is not efficient, because it crowds the memory. Most efficient is to simply use the same variable name and overwrite the previous data after each step. However using different variable names allow the user to independently proof the steps of the process. This makes the process more interpretative, because it is more transparent how the data is transformed from one step to the next. Table 2 shows the variable names applied to the train data during different steps of the process.

Table 2: Variable Names

Variable Name	Process-Step
train_texts_l	Loaded Data
train_texts_p	Pre-Processed Data
train_texts_s	Splited Data
train_texts_b	Sequenced Data
train_texts_bm	One-Hot-Encoded Data
train_texts	Padded Data

4.2 Pre-Processing Sensitivity Study

A sensitivity study was conducted to determine the impact of pre-processing on the computational run-time as well as on the accuracy of the resulting prediction. Minimal pre-processing such as lowercasing, removing of punktuation and of non-english language characters was used as a baseline. Further pre-processing attempts such as stemming and stop words removal either decreased the accuracy or increased the computational time without making the prediction better. Table 3 shows the results of the sensitivity study. It was interestingly found that using NLTK's removal of punctuation takes longer than using the regular expressions module, about three times longer as seen in Table 3. This is, because to use NLTK, tokenization is first necessery, which is executed in a loop. Without the loop, NLTK will default to character level tokenization, for which the models in this project are not optimized. In contrast, the REGEX module performs matching, removal and replacement of punctuation characters, without looping through the text.

Table 3: Pre-Processing Study
(Sample Size: 50k train, 10k test, Vocab Size: 2000)

Pre-Process Operation	Run-Time (Min.)	NN w/ Embed. (%)
Punktuation and non-ASCII, REGEX (baseline)	0.56	87.2%
Punktuation and non-ASCII, NLTK	1.56	87.0%
No Pre-Processing	0.49	87.3%
Punktuation, non-ASCII, Stop words, Stemm, NLTK	2.82	86.0%

The results show that even with **no** pre-processing the prediction accuracy of all three models is just as good as when pre-processing was performed. Only the neural network with embedding layer is shown in Table 3, but the performances of the other models is similar. The reason for the good performance of the no pre-processing is due to the word embedding (both the one-hot-encoded and the learned one). The embedding allows the three models to differentiate between the tokens without getting confused by punctuation, stop words or even upper cases. Although one-hot-encoded embeddings don't actually learn linguistic differences between words in contrast to the embedding layer, they seem to perform just as well.

4.3 Vocabulary Size Sensitivity Study

The size of the vocabulary was also varied to determine the sensitivity of the models to it. Decreasing the vocabulary size reduced computational time. The accuracy of the models remained high up to a point where the vocabulary was reduced too much so that the model no longer hat all necessary words to accurately predict sentiment. The results expressed in Table 4 show that even with a small vocabulary of 2,000 compared to the original 64,191 unique words in the corpus, the sentiment analysis can still be accomplished with a decent 87% accuracy. First when the vocabulary goes under 500 words, performance starts to decrease at a greater rate. Run-time decreases with the size of the vocabulary.

Table 4: Vocabulary Sensitivity Study
(Sample Size: 50k train, 10k test, Pre-Process: baseline)

Vocabulary Size	NN w/ Embedding	NN w/ One-Hot	LogReg	Run-Time (Min.)
12,000	88.1%	88.5%	87.4%	1.24
4,000	87.6%	87.7%	87.3%	0.68
2,000	87.0%	87.4%	87.1%	0.54
500	83.1%	83.3%	83.5%	0.50
50	69.1%	66.1%	68.8%	0.43

*Total words in corpus: 64,191

5 Conclusion

5.1 Results Discussion

The results of the pre-processing sensitivity study show that in order to achieve better than 90% prediction, one needs to design more complex neural network architecture rather than implement more pre-processing. The results of the sample size and data encoding study study showed that performance between a one-hot-encoded embedding an a dense embedding is about the same. This is most likely, because the question to be answered is very simple: if a review is positive or negative. Most likely there are only a few words that contribute to tipping the probability towards positive or negative. This was confirmed in the vocabulary size study which showed that even with a small vocabulary the sentiment analysis can still be accomplished with a decent near 87% accuracy. If the assignment was more complex: with multiple classes, some of which with very few observations, maybe the learned embedding would show significantly better results than the one-hot-encoded embedding. The findings of the studies are useful when designing deep learning algorithms in practice. Such algorithms process millions of observations and have to consider computational and storage limitations. A successful algorithm makes trade offs to optimize the desired characteristics of computational feasibility, but also accurate prediction.

5.2 Answering the Question

This project sought to determine if a machine learning model can accurately predict the sentiment of the reviews and if so under what circumstances such as level of pre-processing, data encoding structures, vocabulary size and computational time. It was found that a learned word embedding is a very efficient way to structure data, which allows for scaling it to a large data set. The project bordered the deep learning field due to the encountered problems such as the out of memory issue and the necessity to structure data in a dense form.

5.3 Positives & Negatives of Using ML Methods

A positive of the machine learning approach using the Keras API is the practical application of machine learning as it is done in the real world, where big data tends to be the rule. A negative of this approach is that it opens more complexities to the amateur user. For example, in order to pass the data through a neural network architecture, one is forced to think on a tensor-level. This is necessary in order to comprehend how the data is transformed by the layers.

Appendix A: Python Code

The next page presents the python code used in the project.

References

[1] Bittlingmayer, A. (Last updated 2019-11-18): *Amazon Reviews for Sentiment Analysis.* Available at: `https://www.kaggle.com/bittlingmayer/amazonreviews/metadata`, retrieved on 15.01.2021.

[2] Chollet, F. (2017): *Deep Learning with Python.* Manning Publications; 1st Edition.

[3] Google Developers, (Last updated 2020-02-10): *Embeddings: Translating to a Lower-Dimensional Space.* Available at: `https://developers.google.com/machine-learning/crash-course/embeddings/translating-to-a-lower-dimensional-space`, retrieved on 12.03.2021.

[4] Kaggle Discussion Board, (Last updated 2018): *Sentiment Analysis with Amazon Reviews.* Available at: `https://www.kaggle.com/bittlingmayer/amazonreviews/code?datasetId=1305&sortBy=voteCount`, retrieved on 15.01.2021.

[5] Krause, M. (2021), *Lecture: Neural Networks* [Powerpoint slides]. Machine Learning Methods in Economics, University of Hamburg, delivered Winter Semester 2020/21.

[6] Krause, M. (2021), *Lecture: k Nearest Neighbors, Linear Models, and the Bias-Variance Tradeoff* [Powerpoint slides]. Machine Learning Methods in Economics, University of Hamburg, delivered Winter Semester 2020/21.

[7] Mueller, A., & Guido, S. (2016): *Introduction to Machine Learning with Python.* O'Reilly Media, Sebastopol 1st edition.

[8] SAS Insights, (2021): *Analytics Insights: Machine Learning: What it is and why it matters.* Available at: `https://www.sas.com/en_be/insights/analytics/machine-learning.html#:~:text=The%20iterative%20aspect%20of%20machine,reliable%2C%20repeatable%20decisions%20and%20results.&text=The%20essence%20of%20machine%20learning.`, retrieved on 12.03.2021.