CS985: Machine Learning for Data Analytics

Goodreads Rating Classification Task.

Overview:

Goodreads Rating Classification Task Notebook.

In this project, we built a deep learning model that is used to predict the review rating of a book review that ranges from 1 to 5. We used the Goodreads dataset that contains reviews from Goodreads which is a website for book reviews. It contains 900000 book reviews from about 25,475 books and 18,892 users.

We started off by preprocessing the data by removing unnecessary characters like punctuation, non alphabetical characters, urls, newline characters and stopwords.

Next, we built our deep learning models using Naive Bayes, LSTM, 1DCNN and GRU. Naive Bayes being the standard ML baseline. A 3 layer NN, a deep NN which extends the 3 layer NN. The complex models we built include LSTM, 1DCNN and GRU.

Overall we found that 1DCNN was the best performing model. Which is the model we would recommend for this use case.

We learned that preprocessing is an important step while working with NLP.

Method:

In this project, the methodologies used for modeling involve several deep learning models including Naive Bayes, a 3 layer NN, a deep NN, LSTM, 1DCNN and GRU. These models were trained and tested on the Goodreads dataset which contains data regarding book reviews. Before modeling, the data was explored and preprocessed. The dataset includes:
user_id - ID of the user
book_id - ID of the Book
review_id - ID of the review
rating - rating of the book with range from 1 to 5
review_text - text that contains the book review
date_added - date of when the review was added
date_updated - date of when the review was updated
read_at - read at
started_at - started at
n_votes - number. of votes
n_comments - number of comments

The columns review_text and rating are the most important columns. The rating would be the target. The review_text is the column that will provide most information about the book. We used Natural Language Processing on this review_text.

The data was then preprocessed by removing unwanted data such as punctuation, urls, non alphabetic characters, spoiler alerts characters and next line characters using the regular expression module. All the text is then converted to lowercase. And the duplicates are dropped.

Using the NLTK library we import a set of stopwords. Stopwords are words that do not add meaning to sentences. Stopwords are common words that are usually excluded during text analysis because they do not have significant meaning.

We then use the TfidfVectorizer from the scikit-learn library to create a matrix of TF-IDF values for 'review_text'.

We perform dimensionality reduction on a sparse matrix using the TruncatedSVD. The csr_matrix method is used to convert the TF-IDF matrix to compressed sparse row format to efficiently store large sparse matrices.

We perform one-hot encoding on the target column using the to_categorical method from keras. This is used to convert categorical data into a format that can be used to target variables.
We subtracted 1 from the rating numbers as tensorflow prefers labels starting from 0, this ensured the number of classes were 5 and not 6. We reversed this when predicting on the kaggle test data by adding 1 to the predictions.

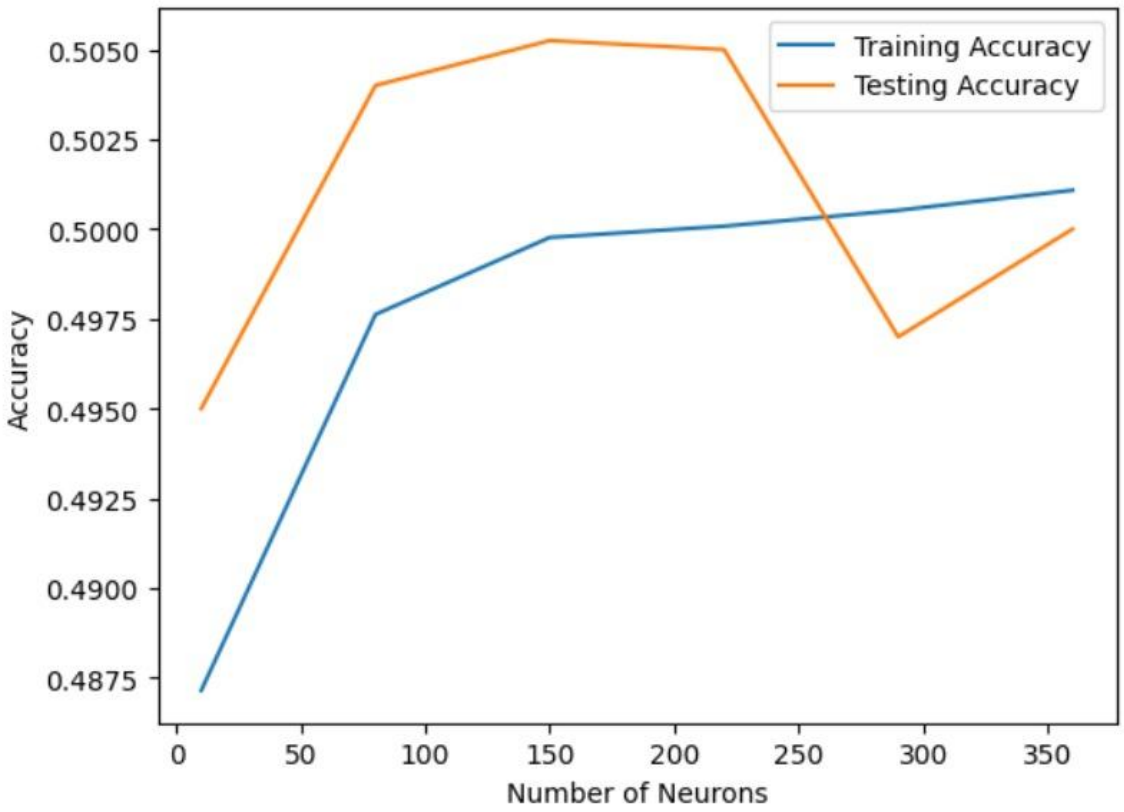
The dataset was then split into train and test data before fitting the model, with 85% of data used for training and the rest was used for testing.

Models:

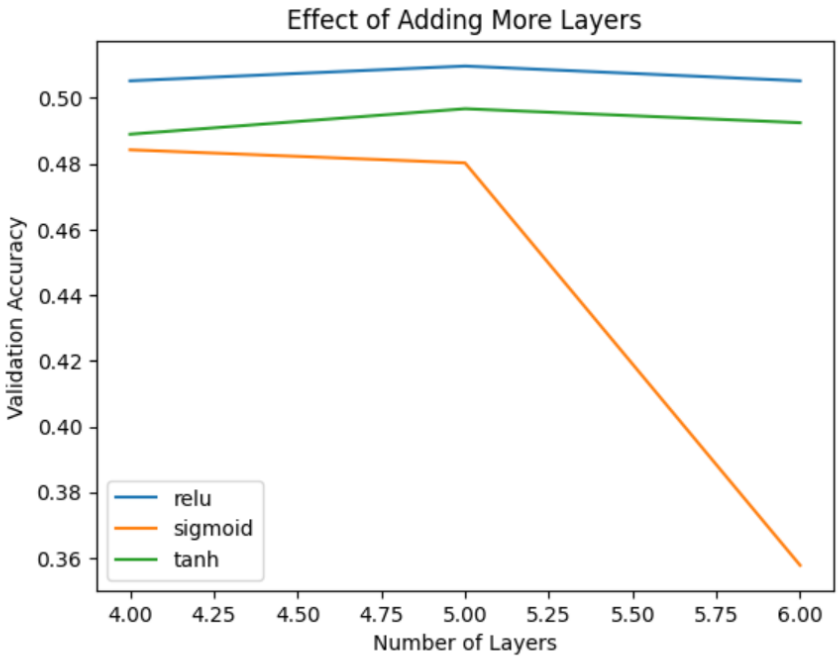
Standard ML Baseline: A simple ML baseline model that uses Gaussian Naive Bayes algorithm from the sklearn library. The model is fit and is used to predict the rating. This model has an accuracy score of 0.385.

3Layer NN: We built a three layer architecture that consists of two dense hidden layers and one dense output layer. The input to the model is a 1x50 dimensional tensor. The first hidden layer has 128 neurons and uses the ReLU activation function. The second hidden layer has 64 neurons and also uses the ReLU activation function. The output layer has 5 neurons and uses the softmax activation function. The model is compiled using the categorical cross-entropy loss function, the Adam optimizer, and accuracy as a metric for evaluation.

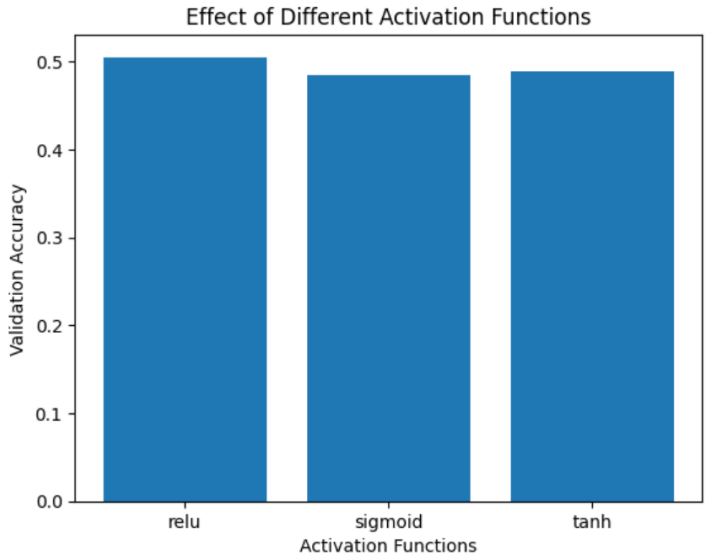
The following graph shows how adding more neurons changes performance



Deep NN models: For our deep NN model we extended the baseline. The model is composed of a Flatten layer that flattens the input, a sequence of dense layers with the set number of hidden units and activation function, and a softmax activation function that provides a probability distribution over the output.



Deeper in this case is not better since the validation accuracy goes down after the 5th layers. This is due to overfitting in the case of relu and sigmoid functions. The large drop in the model with the sigmoid activation function is clearly due to the vanishing gradient problem.



For this task, relu activation function works the best.
We could have tried newer activation functions such as the ‘swift’ function produced in 2017.

Complex NN models: For complex models we used 1DCNN, LSTM and GRU.

1DCNN: we used 1DCNN (one dimensional neural network). 1DCNN is a neural network used for processing text data. We used 1DCNN with glove embeddings. 1DCNNs are an effective tool

for NLP tasks because they can detect local patterns in word sequences while being easy to train and resistant to noise. They have fewer parameters than traditional fully connected networks, which makes them more efficient to train and less prone to overfitting.
GloVe embeddings are especially useful for NLP tasks because they capture the meaning of words in a dense vector space. Words with similar meanings will have similar vector representations, which is useful for capturing semantic relationships between words. By using GloVe embeddings as input to a 1DCNN, the network can learn to recognize patterns in the text that are related to the meaning of the words. We used the Glove embeddings with 50-dimensional word embeddings because of low computational power, we got a higher accuracy when using higher dimensions on Google Collab Pro.

We used Glove embeddings for 1DCNN and for every other model used a TfidfVectorizer and TruncatedSVD to convert the TF-IDF matrix to compressed sparse matrix.

The layers include:

- Embedding layer: This layer maps the discrete word indices to vector representations of size 50. The embedding matrix is initialized with pre-trained GloVe embeddings, and the layer is set to be non-trainable.
- Convolutional layers: The model is enhanced with two convolutional layers each containing 64 size-3 filters, ReLU activation, and the "same" padding. The word embeddings are subjected to filters in these layers in order to identify any local relationships between neighboring words.
- MaxPooling1D layer: This layer down-samples the feature maps generated by the convolutional layers by taking the maximum value over a window of size 2. This helps to reduce the number of parameters in the model and improve its generalization ability.
- Dropout layer: This layer randomly drops out 30% of the neurons in the previous layer during training, which helps to prevent overfitting and improve the robustness of the model.

LSTM: We used LSTM (Long Short Term Memory) for one of our complex neural networks. LSTM is a type of RNN. We implement LSTM by importing Sequential, LSTM, Dense, Embedding and the Adam Optimizer.

- The first layer is LSTM with 5 neurons, 'input_shape' parameter is set to (1, 50), which corresponds to the reshaped input data for the LSTM, and the activation factor is set to 'tanh'.
- The second layer is the BatchNormalization layer. This layer applies a transformation to the input data that ensures that the outputs of the previous layer have zero mean and unit variance, which can help with the training of the model.
- The third layer is a Dropout layer with a rate of 0.3. Dropout is a regularization method that, during training, randomly removes some of the outputs of the preceding layer. This can assist in avoiding overfitting.
- A Dense layer with five units and a softmax activation function makes up the fourth layer. The output of the model is a probability distribution over the 5 potential ratings, which is produced by this layer.

- This model has an accuracy of 0.462141.

GRU: We created a neural network model that consisted of three layers of Gated Recurrent Units (GRU). GRU is a type of RNN model. Each GRU layer has a different number of units (64, 32, and 16), and uses the tanh activation function. The final GRU layer returns a single output, which is then passed through the Dense layer with softmax activation to produce the final output of the model, which is a probability distribution over the 5 possible output classes.

Result:

Model	**Accuracy**	**Mean**	**Standard Deviation**
Gaussian Naive Bayes	0.385	0.35	0.035
3 layer NN	0.5063	0.5001	0.0062
Deep NN	0.5098	0.5052	0.0046
1DCNN	0.514	0.508	0.0061
LSTM	0.492	0.486	0.0043
GRU	0.497	0.487	0.0056

Provide a short summary of the findings given the performance you obtained, and what you tried, and what you learnt.
Explain what you tried, what worked, and what didn't.

From the above table we find that the 1DCNN has the highest accuracy. We tried various models and learned the importance of preprocessing text for NLP. We also realized how hard it is to handle text data because of the large sparse matrices it creates. We were not able to handle the vectorized text data on google collab.Truncated singular value decomposition of sparse matrices was a method we learnt through trial and error that seems effective for such problems. The pre-trained glove embeddings were clearly the differentiating factor as shown in the 1D CNN case. Hardware limitations are a major problem as using 300 dimension (instead of 50) embeddings on the 1D CNN model on google collab pro produced an improvement of around 7 percent accuracy on the kaggle dataset.

Summary:

Which model do you recommend and why?
1DCNN is the model we recommend. We implemented 1DCNN with glove embeddings and it gave us the highest accuracy. Our approach of using 1DCNN with GloVe embeddings was a powerful approach and the combination of these two gave us high accuracy. We used the Glove embeddings with 50-dimensional word embeddings because of low computational power, we got a higher accuracy when using higher dimensions on Google Collab Pro.

How well did it perform on Kaggle?
On Kaggle 1DCNN with Glove embeddings got an accuracy of 58.741%.

What can you do next?
Increase the number of dimensions of Glove embeddings
Increase length of padded sequences
Optimise hyperparameters (Bayesian optimization is recommended)
We can add more layers to this model

References:

regex101: build, test, and debug regex. [online]. Available at: https://regex101.com/

Keras: Deep Learning for humans. [online]. Available at: https://keras.io/

API Reference — scikit-learn 1.2.2 documentation. [online]. Available at: https://scikit-learn.org/stable/modules/classes.html

Guide | TensorFlow Core. [online]. Available at: https://www.tensorflow.org/guide

Hugging Face - Documentation. [online]. Available at: https://huggingface.co/docs

GloVe: Global Vectors for Word Representation. [online]. Available at: https://nlp.stanford.edu/projects/glove/