Multilayer Perceptron for Classification of Movie-Related Utterances

Shannon Rumsey
University of California, Santa Cruz / Baskin School of Engineering
sarumsey@ucsc.edu

Abstract

Feature engineering and hyperparameter tuning play a crucial role in machine learning algorithms. In constructing a model to predict relations for a given movie-related utterance, feature engineering proved to be an effective method for improving accuracy. The usage of dense vector embeddings, batch data processing, data shuffling, and data resampling increased validation accuracy by 3%. Hyperparameter tuning, such as increasing the learning rate and decreasing regularization, boosted validation accuracy to 96% and test accuracy to 81%.

1 Introduction

The objective of this project is to create a model that will determine knowledge graph relations from movie-related utterances, in accordance with the Freebase schema. The Freebase schema is a method used by the Freebase database where an utterance, also called a topic, has relations associated with it, also known as types. For consistency, utterances will refer to topics, and relations refer to types.

Two datasets are considered: a training dataset with utterances that are labeled with relations, and a testing dataset with utterances that do not have associated relations. The training set includes three columns: `"ID"` for the numerical identification of the row, `"UTTERANCES"` for the movie-related utterances a user may input into the model, and `"CORE RELATIONS"` for the relations associated with the utterance (see Table 2). The test data contains only the `"ID"` and `"Utterances"` columns (see Table 3), and the model output is the predicted relations associated with each utterance from the test data. There are a total of 19 possible individual relations (see Table 1), however, there can be multiple relations assigned to an utterance, making this a multi-label classification problem. A relation of note is none because it is both a learnable relation and a placeholder if no relations are assigned to an utterance.

Core Relations
actor.gender
gr.amount
movie.country
movie.directed_by
movie.estimated_budget
movie.genre
movie.gross_revenue
movie.initial_release_date
movie.language
movie.locations
movie.music
movie.produced_by
movie.production_companies
movie.rating
movie.starring.actor
movie.starring.character
movie.subjects
none
person.date_of_birth

Table 1: All unique relations found in the training dataset

The (supervised) model architecture of interest for this task is a multilayer perceptron, which takes an utterance as input and outputs one or more relations associated with the utterance.

2 Models

Two variations of a multilayer perceptron (MLP) model are explored. While the architecture and hyperparameters of the two models differ slightly, the main difference is in data processing techniques.

2.1 Initial Model

The first variation of the MLP model has 3 layers, with 0.2 dropout and a GELU activation function applied to each layer. It uses a weight decay value

ID	UTTERANCE	CORE RELATIONS
0	star of twilight	movie.starring.actor
1	who is chewbakka	movie.starring.actor movie.starring.character
2	find some suspenseful movies	movie.genre

Table 2: Example columns from the training dataset.

ID	UTTERANCE
0	star of thor
1	who is in the movie the campaign
2	list the cast of the movie the campaign

Table 3: Example columns from the test dataset.

of 0.01 (via the AdamW optimizer), and a larger learning rate of 0.006 (Xiao, 2024).

This model was trained on 500 epochs and did not utilize any batch processing techniques. All relations that occurred less than 100 times in the training data were resampled to make the relations more evenly distributed. The inputs consisted of the utterance data as a term-frequency matrix and the core relations information as a binary matrix, where each column represents one unique relation.

2.1.1 Embeddings

Term-frequency (TF) embeddings were used to encode the utterance text. This embedding technique is a simple sparse vector representation that counts the occurrence of words in each utterance. The advantage of using such an embedding is that it is computationally inexpensive, enabling faster training when tuning hyperparameters.

Manual inspection of the model outputs suggests that this type of embedding is not complex enough to support entity recognition or capture semantic meaning. For example, the model struggles with utterances such as `"who did twilight"`, where the meaning of "did" is ambiguous because it can refer to acting, directing, or producing. Scikit-learn's CountVectorizer class was used to implement term-frequency (TF), and the `MultiLabelBinarizer` class was used to create the binary matrix.

2.2 Final Model

The second variation of the MLP model has 4 layers and a GELU activation function applied to each layer. The dropout from the first variation of the model was removed. Implicit weight decay of 0.01 is introduced with the AdamW optimizer. The learning rate remains the same, with a value of 0.006.

Early-stopping was implemented to prevent overfitting from training on excess epochs. Although the maximum number of epochs that the model can train on is 150, the final model converges at around 80 epochs. Utterances were converted into term-frequency inverse document frequency (TF-IDF) and GloVe embeddings, which were then concatenated into one input. The core relations were encoded as a binary matrix, similar to the initial model. The resampling threshold remains the same as the initial model, at 100, but batch processing and data shuffling were introduced.

2.2.1 Embeddings

Term frequency-inverse document frequency, or TF-IDF, is a modified version of the term-frequency embeddings. Term-frequency embeddings measure the frequency at which a term appears in the dataset, whereas term frequency-inverse document frequency (TF-IDF) builds on this by also penalizing common words. This improves upon the original embedding by placing more emphasis on descriptive words, such as the names of movies and actors, instead of frequent words like "the".

In addition to the TF-IDF embeddings, Stanford's GloVe pre-trained twitter-200 embedding model (Jeffrey Pennington, 2014) was also used. The `glove-twitter-200` model has larger vector sizes, which allows for more features to be captured. The content of the data, Tweets, resemble the utterances from the training data better when compared to the other GloVe model trained on Wikipedia data. Unlike TF-IDF and TF embeddings, these dense vector representations preserve named entity recognition and semantics better (Jeffrey Pennington, 2014). For these GloVe-derived embeddings, out-of-vocabulary (OOV) utterances were replaced with vectors filled with random numbers, as this provides more meaningful infromation to the model than vectors of zeros. Because this type of embedding is applied sequentially to the utterance data, using the average of the embeddings for the OOV utterances may bias the embedding towards the data points before it.

Scikit-learn's TfidfVectorizer class was used to implement TF-IDF, and the MultiLabelBinarizer class was used to create the binary matrix. Gensim's downloader was used to load the pre-trained Twitter-200 GloVe embeddings.

While the initial model struggled with the utterance `"who did twilight"`, this model was able to assign the labels `"movie.starring.actor"` and `"movie.starring.character"`. The ambiguity of the word "did" still exists, though the model is now selecting a meaning for it (in this case "did" refers to "acting"), as oppose to labeling it as `"none"`.

3 Experiments

The training dataset was split further into a validation and training set, where the validation set is 20% of the data and the new training set is the other 80%. Because the majority of the modeling was done in PyTorch, the implementation of automatic hyperparameter tuning, such as scikit-learn's grid search, is rather unintuitive. For simplicity, the hyperparameter values used in both models were primarily found through experimentation.

3.1 Data Resampling

In both the validation and training dataset, there are 47 combinations of relations assigned to utterances, with their occurrences varying drastically. For example, the relation `movie.starring.actor` classifies 72 utterances, whereas the combination of the relations `movie.production_companies` and `movie.produced_by` classifies only one. Resampling is implemented to mitigate such imbalances in the dataset. All rows where the associated relations occur less than a designated threshold number of times will be pooled and sampled from with replacement until the total number of observations sampled equals the number of observations not being sampled. The highest frequency of a relation, or, combination of relations, is 323, and the lowest is 1, making the data widely distributed. The mean number of occurrences is 38, yet the median is 5, suggesting a right skewed distribution (see Figure 1).

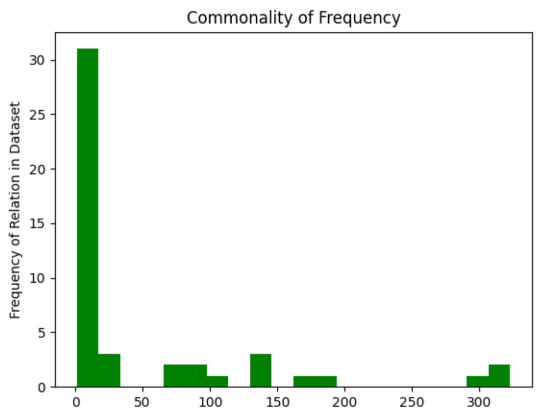

Figure 1: Distribution of the frequency of particular relations.

3.2 Initial Model

The optimizer used is AdamW, an improved version of Adam that contains implicit regularization through weight decay (Loshchilov and Hutter, 2019), which was inspired by the hyperparameters in Guo (2024). Ideally, the learning rate should begin larger, and decrease as it approaches the minimum of the loss function. PyTorch's `ReduceLROnPlateau` learning rate scheduler allows this behavior by lowering the learning rate by a factor of 0.1 any time the loss plateaus for more than 10 epochs.

The initial hyperparameters used in this model were inspired by those used in Guo (2024). These hyperparameters are 300 epochs, 0.001 learning rate, and 0.6 dropout. The hyperparameter values experimented with can be found in Table 4.

3.3 Final Model

This final model uses the same learning rate scheduler and AdamW optimizer as the initial model. Early-stopping is used instead of fine-tuning training epoch lengths. A maximum epoch threshold is used (150), though the actual number of epochs that the model is trained on varies and is typically smaller than this figure.

Changes made between this model and the initial are: the addition of batch training, data shuffling, new input embedding techniques, and one extra layer. While the hyper-parameter changes illustrated in Table 5 impacted accuracy, the main improvements of this model can be attributed to data manipulation and training methods. As inputs, this model uses both TF-IDF and GloVe embed-

Resamp. Threshold	Epochs	Dropout	Learn. Rate
None	300	0.6	0.001
None	500	0.6	0.001
None	500	0.4	0.001
None	500	0.2	0.001
None	500	0.2	0.002
None	500	0.2	0.004
None	500	0.2	0.005
None	500	0.2	0.006
None	500	0.2	0.008
30	500	0.2	0.006
80	500	0.2	0.006
100	500	0.2	0.006
200	500	0.2	0.006
150	500	0.2	0.006

Table 4: Different hyperparameter values used in the initial model.

dings (Jeffrey Pennington, 2014). When training the model, PyTorch's `DataLoader` class was used for batch processing and data shuffling. Each batch had 64 rows from the training data (Xiao, 2024). Shuffling the dataset was enabled because the training dataset is sorted by core relation.

The hyperparameters experimented with can be found in Table 5.

Resamp. Threshold	Dropout	Learn. Rate
100	0.2	0.006
100	0.1	0.006
100	0	0.006
100	0	0.004
100	0	0.008
150	0	0.006
80	0	0.006

Table 5: Different hyperparameter values used in the final model.

3.4 Evaluation & Model Selection Criterion

The evaluation metrics used are validation and test accuracy, and training loss. Accuracy is calculated by averaging the number of utterances that have all of the correct relations predicted. The loss is calculated using PyTorch's `BCEWithLogitsLoss`, which combines Binary Cross Entropy and a sigmoid layer (Guo, 2024).

The initial model was trained for 500 epochs, however, early-stopping was used for the final model. The number of epochs to train the final model was set to a maximum of 150, though the actual amount used to train ended up being around 80 due to the early-stopping that is initiated when two conditions are met: validation accuracy is neither the same nor increasing, and the loss is not decreasing. This stops once the highest accuracy achieves the lowest loss, which prioritizes accuracy, but also ensures that the accuracy is not a result of overfitting. Early stopping resulted in roughly a 0.001 increase in validation accuracy (see Table 6).

Epoch Stopped	Loss	Val. Acc.
82	$1.337300e^{-6}$	0.973936
150	$5.714579e^{-7}$	0.972565

Table 6: Validation accuracy and loss differences between early stopping and completion of epochs.

4 Results

The most substantial impact on accuracy and validation loss was due to feature engineering. The findings from Aydoğan and Karci (2020) and Mandelbaum and Shalev (2016) confirm the importance of embeddings for accuracy in a classification-tasked neural network. Term frequency-inverse document frequency is also considered a better option for embeddings compared to bag-of-words embeddings (Stephen et al., 2022). The initial and final model share many similarities, however the validation accuracy and loss improved by 0.03 when batch processing, data shuffling, and high dimensional embeddings were implemented.

Other than feature engineering, the final model implemented zero dropout, the only regularization was weight decay in the AdamW optimizer. The work done by Hernández-García and König (2018) suggests that dropout is not needed if implicit regularization, such as the weight decay in the AdamW optimizer, is used. The larger learning rate is supported by the findings in Lobacheva et al. (2023).

Model	Loss	Val. Acc.	Test Acc.
Initial	0.000423	0.936899	76.758%
Final	$5.048639e^{-6}$	0.963704	81.549%

Table 7: Validation accuracy and loss differences between early stopping and completion of epochs.

4.1 Initial Model

For the initial model, the first experiment resulted in poor accuracy, not much better than a coin toss. However, after improving the number of epochs, the accuracy increased 20%. The accuracy seems to plateau around 450 epochs during training, therefore the number of epochs remains unchanged throughout the rest of the hyper-parameter experiments. Due to resulting increases in loss and validation accuracy after lowering dropout, dropout was lowered further. The model's performance improved with each increase in the learning rate, up until 0.006.

The last hyperparameter explored was the threshold at which data would be resampled. Because many of the possible relation outcomes had a frequency of 1, very low count thresholds were explored. Higher resampling thresholds generally resulted in better model performance, until a threshold of 200. Interestingly, at this threshold, the model failed to resample all uncommon relations, causing the number of unique individual relations in the final balanced dataset to be less than 19. This threshold introduces too many combinations of relations for the model to select from, making each combination less likely to be chosen, thus causing some to not be chosen at all.

4.2 Final Model

The first hyperparameter altered was dropout. Lowering dropout led to an improvement in accuracy, however, removing dropout entirely achieved the best performance in terms of both loss and validation accuracy. Changing the learning rate and the resampling threshold had no significant impact on results.

References

Murat Aydoğan and Ali Karci. 2020. Improving the accuracy using pre-trained word embeddings on deep neural networks for turkish text classification. *Physica A: Statistical Mechanics and its Applications*.

Yanming Guo. 2024. Multimodal multilabel classification by clip.

Alex Hernández-García and Peter König. 2018. Do deep nets really need weight decay and dropout?

Christopher D. Manning Jeffrey Pennington, Richard Socher. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Ekaterina Lobacheva, Eduard Pokonechny, Maxim Kodryan, and Dmitry Vetrov. 2023. Large learning rates improve generalization: But how large are we talking about? In *NeurIPS 2023 Workshop on Mathematics of Modern Machine Learning*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

Amit Mandelbaum and Adi Shalev. 2016. Word embeddings and their use in sentence classification tasks.

Akuma Stephen, Tyosar Lubem, and Isaac Adom. 2022. Comparing bag of words and tf-idf with different models for hate speech detection from live tweets. *International Journal of Information Technology*, 14.

Lechao Xiao. 2024. Rethinking conventional wisdom in machine learning: From generalization to scaling.

A Hyperparameter Values

Table 8 shows the hyperparameter values used to tune the initial model and Table 9 shows the values used in the final model.

Resamp. Threshold	**Epochs**	**Dropout**	**Learn. Rate**	**Loss**	**Val. Acc.**
None	300	0.6	0.001	0.061847	0.596112
None	500	0.6	0.001	0.0326476	0.779697
None	500	0.4	0.001	0.0114300	0.827213
None	500	0.2	0.001	0.006325	0.831533
None	500	0.2	0.002	0.003750	0.838012
None	500	0.2	0.004	0.003186	0.838012
None	500	0.2	0.005	0.001631	0.838012
None	500	0.2	0.006	0.001050	0.846652
None	500	0.2	0.008	0.001789	0.842332
30	500	0.2	0.006	0.001145	0.923166
80	500	0.2	0.006	0.000998	0.929912
100	500	0.2	0.006	0.000423	0.936899
200	500	0.2	0.006	ERROR	ERROR
150	500	0.2	0.006	0.001337	0.877394

Table 8: Hyperparameter values and their results in the initial model.

Resamp. Threshold	**Dropout**	**Learn. Rate**	**Loss**	**Val. Acc.**
100	0.2	0.006	$1.396678e^{-6}$	0.961591
100	0.1	0.006	8.355602^{-6}	0.973936
100	0	0.006	$1.337300e^{-6}$	0.973936
100	0	0.004	$6.772495e^{-7}$	0.964334
100	0	0.008	$4.038076e^{-7}$	0.968449
150	0	0.006	$1.131872e^{-6}$	0.902298
80	0	0.006	$5.048639e^{-6}$	0.963704

Table 9: Hyperparameter values and their results in the final model.