Multi-class Multi-label Classification on Film-Related Question-Utterances

Darian Lee
UC Santa Cruz
Sunnyvale, CA, 94087
`daeilee@ucsc.edu`

Abstract

This paper examines the strengths and weaknesses of various models for the multi-label classification of film-related utterances. I conclude that the most effective preprocessing method involves concatenating character trigrams with mean-pooled `glove-twitter-200` semantic embeddings while ignoring out-of-vocabulary (OOV) tokens. The final model architecture consists of four hidden layers, each containing 1,500 neurons, and utilizes the leaky ReLU activation function to address the dying ReLU problem observed in previous iterations. The model achieved a final testing accuracy of 0.81549. I believe obtaining a significantly higher accuracy on this dataset is unlikely due to noisy labels and insufficient training data for certain classes.

1 Introduction

The goal of this project was to build a multi-class multi-label classification model capable of categorizing film-related utterances by topic. The nineteen classes the model sought to predict included `actor.gender`, `gr.amount` (the film's grossing in comparison to other films), `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`, `person.date-of-birth`, and `none`. These classifications were made on a column of question utterances ranging from 1 to 21 words.

This project utilized supervised learning on a labeled dataset sourced from Kaggle, authored by unknown contributors. The class identities were somewhat imbalanced, predominantly represented by the starring.actor and movie.directed-by class, while utterances related to birthdays, music, and seven other classes constituted only 3.3% of the dataset combined (Figure 1). This project was motivated by a desire to improve the efficient retrieval of film information based on specific queries and has applications in film recommendation systems, customer support, and other similar domains.

UTTERANCES	CORE RELATIONS
i need to get a list of some romantic indian movies	movie.country movie.language movie.genre

Table 1: Example row (row 1000 in dataset)

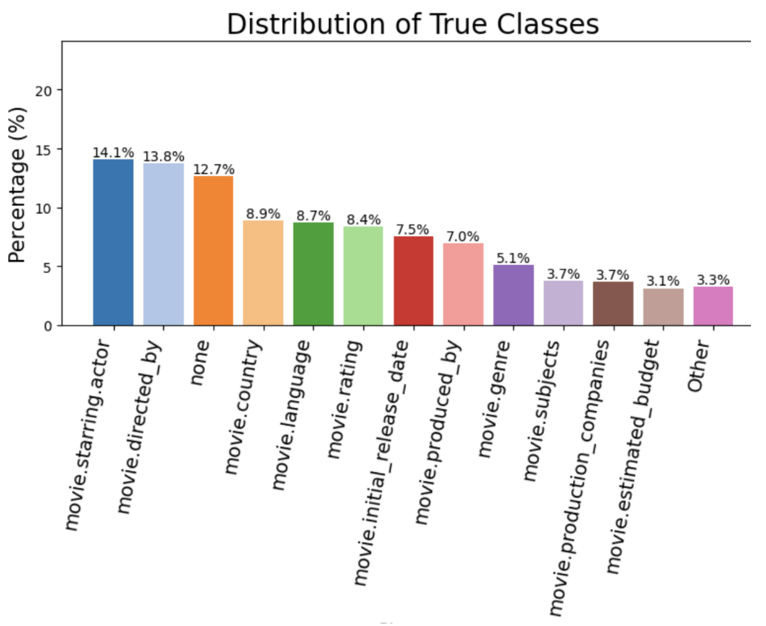


Figure 1: Distribution of class identities in the training data, showing the percentages of each class represented. The 'Other' category includes all classes that fall below the top 12 in frequency.

2 Models

In this section, I will outline the architecture and motivations behind my three most distinct models. While I evaluated many additional models, I have chosen to concentrate on these three because they each showcase unique and innovative approaches to addressing the problem based on scientific research. The other unmentioned models were similar variations and combinations of these approaches. I will start with a brief description of the "base model", which can be interpreted as the common aspects shared by each of these models. I will reserve the discussion of the final model for later in the experiments and results section to illustrate the progression that led to its development.

2.1 Baseline model

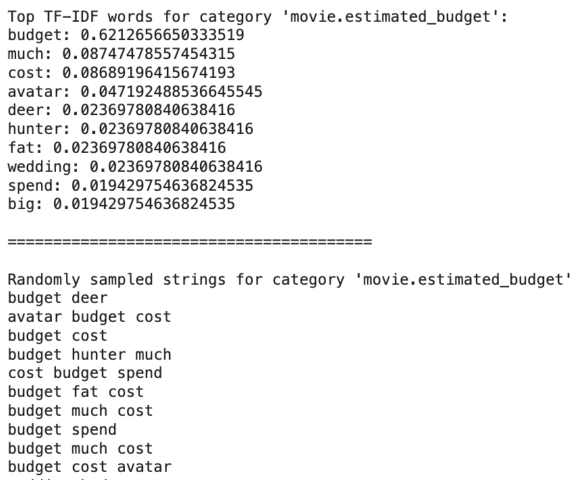
The text data was lemmatized for all models, and stop words were removed. Some version of a pre-trained semantic embedding vector was then fit on the tokenized utterances and mean-pooled to ensure utterances with different token lengths had the same dimensions. The most common embedding I used was `glove-twitter-200` due to the informal style of the dataset and its high testing accuracy in experimentation, however, I also tried versions of `glove-wiki-gigaword` or combinations of both embeddings concatenated together. OOV tokens were initially handled with a vector of zeros. Still, upon further consideration of the effect that would have on the pooled mean of the utterance, I opted for emitting OOV tokens from the embeddings entirely. In most of my highest accuracy models, these embeddings were concatenated with a normalized bag-of-words representation of the text, either containing full words, word n-grams, or character n-grams. The class identities were transformed into a sparse matrix with 19 columns, with a one representing class identity and a zero representing lack thereof. The resulting X matrix was then passed into a custom model in PyTorch with around five layers and anywhere from 200 to 2000 neurons per layer. Different activation functions were tried, although ReLU and leaky ReLU obtained the best results and thus were used predominantly. The optimizer Adam with a 0.0006 learning rate was also used primarily, as it led to the best performance and was supported by a literature review. The training was done in batches of 32 using K-fold cross-validation to assess the model's generalization capabilities on unseen data. Batches were shuffled before each training epoch, and validation accuracy, loss, F1, precision, and recall were all recorded and used as criteria for model selection.

2.2 Model 1: Synthetic Data Creation for Underrepresented Classes

The first major deviation from the baseline model involved incorporating synthetic data to remedy strong class imbalances observed in the training data. New utterances were generated using probabilities generated from TF-IDF scores for each class corpus which consisted of an exhaustive concatenated collection of utterances belonging to the same class. TF-IDF is a widely utilized technique in text classification and summarization tasks; and recently, its application in synthetic data generation has been explored as well [4]. I chose to experiment with TF-IDF for generating realistic synthetic data because it effectively highlights the words that are most indicative of specific class identities. This technique can be transformed into a list of probabilities for the most frequent words by dividing each TF-IDF value by the total sum of all TF-IDF values and then sampling words with these probabilities to form new utterances (Figure 2). Additionally, since the text was represented as a bag-of-words encoding, the grammaticality and sensibility of the generated utterances were not critical to model performance. As long as the synthetic data reflects a realistic distribution of words for each class, it should theoretically function similarly to real data. Its inclusion was expected to improve model performance, particularly for underrepresented classes.

2.3 Model 2: Handling Incorrect Spelling with Character Trigrams and Levenshtein Distance

The second major adjustment stemmed from my desire to avoid data loss due to incorrect spellings and unknown tokens. A review of the literature revealed the benefits of using character n-grams to handle spelling mistakes. In particular, the article *Words versus Character n-Grams for Anti-Spam Filtering* discusses the effectiveness of this method when combined with the bag-of-words approach, highlighting its comparatively lower dimensionality than word n-grams and its ability to capture slight spelling deviations since most n-grams remain the same across different spelling variants [2]. I experimented with various n-gram sizes but eventually settled on trigrams, as they yielded the



```
Top TF-IDF words for category 'movie.estimated_budget':
budget: 0.6212656650333519
much: 0.08747478557454315
cost: 0.08689196415674193
avatar: 0.047192488536645545
deer: 0.02369780840638416
hunter: 0.02369780840638416
fat: 0.02369780840638416
wedding: 0.02369780840638416
spend: 0.019429754636824535
big: 0.019429754636824535

========================================

Randomly sampled strings for category 'movie.estimated_budget'
budget deer
avatar budget cost
budget cost
budget hunter much
cost budget spend
budget fat cost
budget much cost
budget spend
budget much cost
budget cost avatar
```

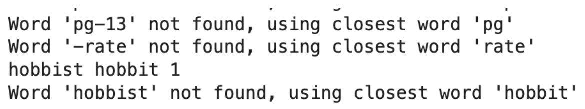
Figure 2: Generated data for the underrepresented class `movie.estimated_budget`. The highest TF-IDF score is for "budget", followed by "much", "cost", and "avatar"– one of the most expensive movies ever made. The generated strings reflect a nonsensical yet realistic distribution of words for this class.

highest testing accuracy.

Additionally, I explored using Levenshtein distance as a method to minimize information loss when handling missing tokens in word embeddings. Initially, I addressed OOV tokens by generating a vector of all zeros. However, I realized that this approach could lead to drastically different pooled means for very similar sentences, depending on the presence of an OOV token. For example, consider the utterances "main actor" and "main actor vulgaria," where "vulgaria" is an out-of-vocabulary word. The element-wise mean of the first two embeddings would be identical; however, including the all-zero embedding for the OOV word in the second utterance would significantly lower the mean, causing the two sentences to appear less similar.

One approach to mitigate this issue is to exclude OOV values from the embeddings entirely, which I began implementing after this realization. However, this method still has the drawback of discarding information from OOV words that are merely misspellings of existing words. Therefore, I decided to use Levenshtein distance to replace OOV words with similar words from the vocabulary, resulting in more informative embeddings (Figure 3).

Although a full implementation proved too time-consuming, I developed a modified version where the model became more lenient about the required



```
Word 'pg-13' not found, using closest word 'pg'
Word '-rate' not found, using closest word 'rate'
hobbist hobbit 1
Word 'hobbist' not found, using closest word 'hobbit'
```

Figure 3: Example of results from using Levenshtein distance for OOV tokens

word similarity the longer it searched. If no word with fewer than four character differences was found after searching 300,000 words, the word was excluded from the embedding. Below is a sample of the function I used.

```
## a modified approach to Levenshtein
    distance where only a fixed number
    of words are searched, rather than
    the full vocabulary
## This approach shrinks the time
    required per token from 2 minutes to
     a few seconds, and results in
    similar output strings
count = 0
def find_closest_word(token, vocab):

    closest_word = None
    smallest_distance = float('inf')

    wordcount = 0
    for word in vocab:

        wordcount += 1
        distance = edit_distance(token
            , word)

        if (distance <= 2) or ((
            wordcount > 90000) and (
            distance <=3)):
          print(token, word, distance)
          return word
        elif (wordcount > 200000) and
            (distance <=4):
          print(token, word, distance)
          return word
        elif wordcount > 300000:
          return None

    return None
```

2.4 Model 3: Excluding None

My third deviation was perhaps the most drastic, and it involved excluding the None class from the training set entirely. The main reasons for this were the relative prevalence of the None class, which could lead to overfitting, and its broad, ambiguous nature, which could introduce noisy decision boundaries. I believed that removing None would allow the model to better focus on the actual target classes and converge more quickly. To still account for None in the test set, I modified my validation rules so that if all the outputs for a given instance were below 0.5, None would be predicted.

3 Experimentation and Intermediate Results

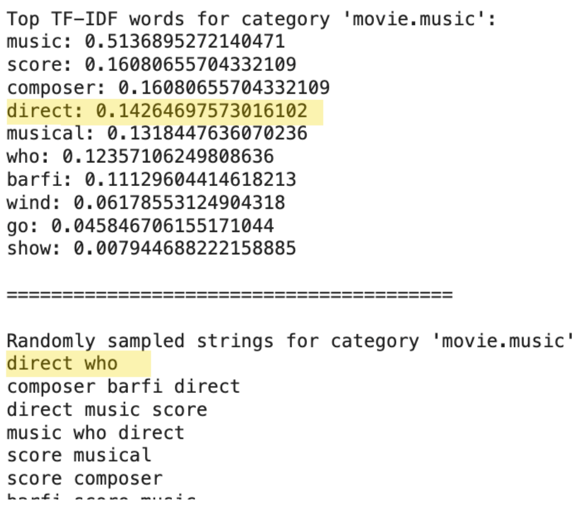
In this section, I will outline the process of going from the 3 model variations described in section 2 to the final model. I will discuss the strengths and weaknesses of each and the changes I made to improve them, as well as the final combination of approaches I chose for my final model.

3.1 Model 1: Synthetic Data Creation for Underrepresented Classes

Unfortunately, upon evaluation, my model trained with synthetic data generated from TF-IDF scores performed worse than similar models with the same preprocessing and parameters but without synthetic data. One likely reason for this decline is the noise introduced in the synthetic data, which appeared in two forms. First, for classes with very few training examples, TF-IDF sometimes assigned high values to words that were not logically associated with the class. For example, in the class `person.date_of_birth`, which only appeared in six utterances in the training set, the generated data included irrelevant strings like `cruise_kristen`. This occurred because there were too few examples to derive meaningful TF-IDF values.

The second source of noise came from potential misclassifications in multi-label instances, especially in underrepresented classes. When a class had few or no standalone examples, I included multi-label utterances in its TF-IDF computation. This led to certain class TF-IDF scores being influenced by words predictive of other classes (figure 4). While I tried to replicate some multi-label patterns based on clear correlations—such as `gr.amount` consistently appearing with `movie.gross-revenue`—it was impractical to generate more complex multi-label relationships due to the limited or nonexistent training data for certain class combinations.

Although alternative methods for balancing classes, such as over- and undersampling, are available, I decided against using them. With some classes appearing fewer than 10 times, undersampling would result in significant data loss. Conversely, oversampling would introduce the same issues as synthetic data generation by repeatedly exposing the model to the same utterances, causing it to overemphasize unrelated words present in those utterances. As a result, I fully abandoned this approach for the final model.



```
Top TF-IDF words for category 'movie.music':
music: 0.5136895272140471
score: 0.16080655704332109
composer: 0.16080655704332109
direct: 0.14264697573016102
musical: 0.1318447636070236
who: 0.12357106249808636
barfi: 0.11129604414618213
wind: 0.06178553124904318
go: 0.045846706155171044
show: 0.007944688222158885

=======================================

Randomly sampled strings for category 'movie.music'
direct who
composer barfi direct
direct music score
music who direct
score musical
score composer
```

Figure 4: Because multilabelled utterances were included in the TF-IDF calculations for `movie.music`, the word "direct" is scored very highly despite having a more apparent relationship with the class `movie.directed-by`

3.2 Model 2: Handling Incorrect Spelling with Character Trigrams and Levenshtein Distance

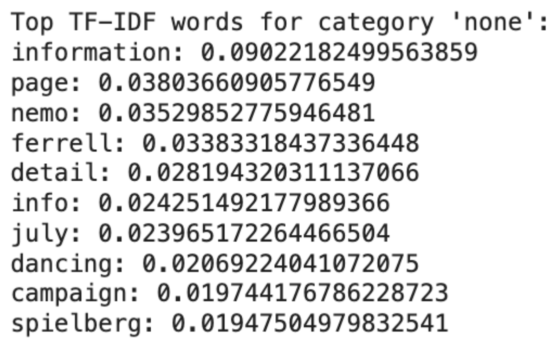
Although the model trained on character n-grams concatenated with embeddings where Levenshtein distance was used for OOV tokens performed better than previous models (0.796 testing accuracy), omitting Levenshtein distance and including only character n-grams raised the testing accuracy to 0.8. I am hesitant to conclude that using Levenshtein distance actively harmed my model, as such a small change in accuracy could be attributed to factors like differences in random initialization, which could have led to the final model being selected after a different number of epochs for different runs (I stopped training when I reached the best accuracy). However, these results suggest that using Levenshtein distance did not actively improve the model either.

One possible explanation is the relatively small number of OOV tokens caused by spelling mistakes. More commonly, OOV tokens were proper nouns that the vocabulary had not seen before. Thus, converting the movie title "Vulgaria" into "vulgar" likely had no effect or even a negative effect on the model's ability to group similar utterances. Another reason is the redundancy of using two different approaches to solve the same problem of misspellings and unknown tokens. If character n-grams were already allowing the model to represent similar spellings with similar vectors, using Levenshtein distance would not add any new information. Therefore, going forward, I opted to include only character n-grams.

3.3 Model 3: Excluding None

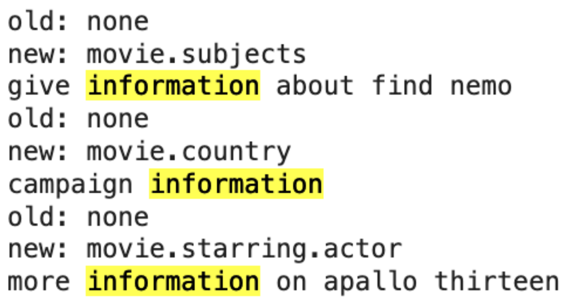
The model from which I excluded the “None” class performed poorly, generating a CSV of predictions so obviously incorrect upon human examination that I did not want to waste a submission on Kaggle. One reason for this might have been my overconfidence in the broad and ambiguous nature of the “None” class. Upon further analysis of the TF-IDF scores for the none class and a review of the training data, I found that many utterances labeled as none shared common words such as “information” and “page,” which did not frequently occur in other classes (Figure 5). It appeared that “None” was more related to finding plot information and summaries of movies, rather than serving as a catch-all for any utterance not related to the other classes as I initially believed. Thus, by excluding the “None” class, my model was losing valuable information. I observed this deficency in the testing csv as it tended to label sentences with words that scored high TF-IDF values in the “None” wrong (Figure 6).

Another reason for the poor performance was that in my efforts to prevent the model from overfitting and predicting the “None” class too often, I ended up creating a model that predicted it too infrequently. Whereas my best model predicted the “None” class 118 times, the model that excluded the “None” class from the training data only predicted it 55 times. I attempted to modify the thresholds to increase the frequency of “None” predictions; however, the CSV file continued to look worse than those of other models, leading me to ultimately abandon this approach as well.



```
Top TF-IDF words for category 'none':
information: 0.09022182499563859
page: 0.03803660905776549
nemo: 0.03529852775946481
ferrell: 0.03383318437336448
detail: 0.028194320311137066
info: 0.024251492177989366
july: 0.023965172264466504
dancing: 0.02069224041072075
campaign: 0.019744176786228723
spielberg: 0.01947504979832541
```

Figure 5: Results show presence of words indicative to None class



```
old: none
new: movie.subjects
give information about find nemo
old: none
new: movie.country
campaign information
old: none
new: movie.starring.actor
more information on apallo thirteen
```

Figure 6: utterances containing the word “information” which were predicted as “none” in previous models were predicted differently and inconsistantly under the new model

3.4 Arriving at my Final Model

After a lengthy experimentation process, I concluded that the best way to preprocess my final model was by tokenizing using character trigrams and concatenating that with the `glove-twitter-200` embeddings of the words, excluding out-of-vocabulary (OOV) tokens from the embeddings. The model consisted of four hidden layers, each containing 1,500 neurons, and utilized the leaky ReLU activation function with a negative slope of 0.01. I decided to switch from ReLU to leaky ReLU because I suspected that the dying ReLU problem was occurring in my dataset, evidenced by rows where every class was predicted as zero [1].

I had employed a decision rule where classes with predictions above 0.5 would be marked as true; if no classes exceeded this threshold, the argmax would be set to true. However, the presence of all-zero rows rendered this rule inapplicable. Leaky ReLU addresses the dying ReLU issue by allowing for slight negative outputs, which is why I chose it.

Additionally, I used the Adam optimizer with `nn.BCEWithLogitsLoss()` because it yielded the best performance. I trained my data using 5-fold cross-validation, with each fold consisting of 25 epochs. The final model was saved as the one with the highest F1 score. I also tested models selected based on the highest accuracy score, the lowest loss score, and a combined metric of accuracy plus F1 minus the square root of loss. However, the model selected based on the highest F1 score performed the best, likely due to the class imbalance in my dataset.

High accuracy can be achieved by frequently predicting majority classes, which can lead to low

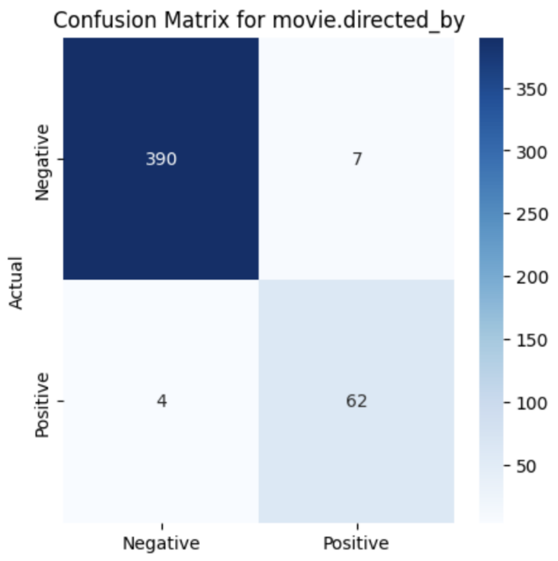


Figure 7: Confusion matrix from validation set, revealing high precision and recall in final model

recall as the number of false positives increases. In contrast, F1 scores take both precision and recall into account, ensuring a balanced evaluation of the model's performance. This approach guarantees that the final model can effectively predict minority classes, which is especially important in imbalanced datasets like the one I was working with.

3.5 Results of Final Model

The final model achieved a testing accuracy of 0.81753 on Kaggle, placing me 10th on the leaderboard for the best model predictions (tied with 4 other people with the same score). As discussed in earlier sections, I believe this success was due to my use of character n-grams to enhance similarity across different spellings, the exclusion of out-of-vocabulary (OOV) words from embeddings, which resulted in more stable pooled means, and the inclusion of leaky ReLU to address the dying ReLU problem. The high testing accuracy was also likely influenced by the selection criteria of prioritizing the highest F1 score, which enabled my final model to predict minority classes more effectively (Figure 7). My final model aligns with several aspects of other models studied for similar tasks, particularly in its use of trigrams [3], the application of F1 score for model selection [2], and the strategy of ignoring word embeddings for missing tokens [2].

3.6 Linked Models

- **Model 1: Generated Data:** https://colab.research.google.com/drive/1K6Kwr9uEWOAyhNEf378FmzD_lTJJ3exP?usp=sharing
- **Model 2: Character trigrams and Levenshtein distance:** https://colab.research.google.com/drive/1OoH9G75rcV6mYvuZf_ncHoWY2uNA8Jgj?usp=sharing
- **Model 3: None excluded:** https://colab.research.google.com/drive/1CnEKVtUsccUyYf1Mc_iIb8AQiBKVOeXZ?usp=sharing
- **Final Model:** https://colab.research.google.com/drive/10y8EZbgsDO65JFGlZacdbhVlNiZ2L5cC?usp=sharing

3.7 Citations

References

[1] S. C. Douglas and J. Yu, “Why ReLU units sometimes die: Analysis of single-unit error backpropagation in neural networks,” in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, 2018, pp. 864–868, doi: 10.1109/ACSSC.2018.8645556.

[2] I. Kanaris, K. Kanaris, D. Houvardas, and E. Stamatatos, “Words versus character n-grams for anti-spam filtering,” *International Journal of Artificial Intelligence Tools*, vol. 16, no. 6, pp. 1047–1067, Dec. 2007. [Online]. Available: https://www.researchgate.net/publication/220160318_Words_versus_Character_n-Grams_for_Anti-Spam_Filtering.

[3] H. H. Mohammed, O. Mazher, B. Ghaleb, and A. Bashier, “Multi-label classification of text documents using deep learning,” in *2020 IEEE International Conference on Big Data (Big Data)*, Dec. 2020, pp. 4681–4689. doi: https://doi.org/10.1109/bigdata50022.2020.9378266.

[4] M. Grootendorst, “BERTopic: Neural topic modeling with a class-based TF-IDF procedure,” Dec. 2022.