Aneesh Ponduru, Neel Avancha, Gauri Sarin
Professor Tahmasebi
Natural Language Processing
18 April 2024

MovieMate: A Personalized Movie Recommendation System

Abstract:

This paper is an introduction for MovieMate, a chatbot designed to be able to provide personalized movie recommendations and detailed movie information to assist in selecting entertainment to watch from a vast number of options. This implementation integrates numerous concepts discussed in CS4120, such as a variation of the BERT Language model, rule-based sentiment analysis, and analyzing word embeddings via cosine similarity. A fine-tuned DistilBert model was utilized, and successfully demonstrated high aptitude in performing entity recognition and intent classification, scoring over 90% in evaluation metrics such as precision, recall, f1-score. For intent regarding information about a movie, the model calls functions which filter an IMDB movie dataset based on the specified entity, and return the requested information. For intent regarding user preference, two models were incorporated: A TF-IDF model designed to return similar movies to a given movie, and a VADER sentiment analyzer designed to read and adjust a movie's rating based on the positive or negative feedback from the user. These models all worked in tandem together for MovieMate to effectively enhance user experience and ensure personalized recommendations.

Introduction:

Despite the proliferation of digital streaming services, viewers frequently find themselves navigating a vast ocean of movie choices without a clear way to personalize or deeply understand these selections. In fact, there is a phenomenon known as choice overload[1], where humans struggle to pick a satisfactory choice when there are a multitude of options presented. Given the explosion of streaming services such as HBO Max, Netflix, Hulu, Disney+, and so many more, consumers have been overloaded with more choices than ever for consumption of entertainment.

Current solutions to help us narrow down this vast pool, including various streaming service algorithms and search tools like Google, fall short in offering truly personalized experiences. While Google provides extensive data, it lacks interactivity and personalization, making it less effective for viewers seeking tailored movie suggestions. Other tools simply just select a random movie from a list of movies, which only aids in performing the selection process for the user, and completely lacks any criteria or prior considerations.

To address these shortcomings, MovieMate has the ability to provide suggestions based on specified criteria, or from past feedback on a given movie. It can recommend a series of movies based on an inputted genre or on the similarity to another movie, and remember previous feedback to provide updated suggestions. As such, MovieMate is able to provide assistance in a way traditionals browsers cannot.

Methods:

The overall movie recommendation system relied on the creation of a few separate models that were then combined together to create MovieMate. The models included:

Entity Recognition and Intent Classification →

To implement a model to perform named entity recognition and intent classification, a PyTorch model class was created to represent the model in which two heads were created. One

[1] Chernev, Böckenholt, & Goodman. “Choice overload: A conceptual review and meta-analysis”

head was created for entity recognition using DistilBertForTokenClassification with the standard DistilBert configuration and the number of labels were the possible BIO tags that could be assigned to each token. The second head was initialized to perform intent classification and uses a linear layer to map the [CLS] token to the number of possible intent labels. The loss function for both intent classification and entity recognition is CrossEntropyLoss, and the loss function for entity recognition includes a parameter to ignore -100 since this is the padding token.

Next, in the forward pass of the model, the outputs of the model and the hidden layers are calculated using input_ids and attention masks. The [CLS] token, which is used for intent classification, was calculated using the last shape of the hidden states. Then, the logits for intent classification are generated by passing in the [CLS] representation from the model to the intent classifier. Secondly, the logits from the outputs are used for entity recognition to determine which of the tokens represent entities that should be labeled. Finally, the loss is calculated for both tasks, and combined to obtain a total loss for the model. All loss values and logits are then returned by the method.

In the training loop, AdamW was used as the optimizer and the best validation loss to infinity. A validation set is used to determine when to save the model. In order to discourage overfitting on the train set, a model is only saved when the loss of the current model is less than the best loss on the validation set. For each epoch, all loss values, the total loss from the intent head, the entity recognition head, and the total loss, are initialized to zero. For each batch in the train loader, after zeroing the gradients, the loss values are calculated by running the forward pass on the model. After all batches in the train loader are run, the average intent and entity recognition loss are calculated and reported for the epoch. A similar process is repeated for the batches in the validation set loader, and the model is only saved if the total validation loss from the epoch is less than the previous lowest total loss on a validation set. This implementation of the model allows the efficient and accurate classification of intent and also the correct recognition of named entities.

TF-IDF and Cosine Similarity →

To implement a model to work with the movies data and retrieve movie recommendations, statistical analysis using TF-IDF scoring and cosine similarity was used. The director, synopsis, genre, and cast columns were concatenated together into one corpus series, and then preprocessed by removing punctuation, stopwords, converting the text to lowercase. After all the data preprocessing was completed, the corpus was fit to the TF-IDF Vectorizer, and the similarity matrix was computed from SKLearns built-in cosine similarity function. From there, the function would return the most similar movies to the given movies based on the cosine similarity function.

VADER Sentiment Analyzer →

In order to incorporate the feedback aspect of the chatbot, a model was needed to perform sentiment analysis. There are a variety of pre-built models for computing sentiment analysis, and choosing the proper one was crucial. The language from the feedback was going to be short, and typically a few words would carry the majority of the overall sentiment of the sentence. As such, the VADER model was the best fitting model for this use case, as it works well with text produced in a social media setting[2], which share similarity in terms of length as well as general discourse utilized by a user's response to a chatbot. Because Vader is able to recognize the impact on sentiment that punctuation makes as well as capitalization, the only preprocessing done was to lemmatize each word. This was done to help generalize rules across different word

[2] Hutto, Gilbert, "VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text"

forms with the same root word (ex: run, running, ran). VADER is able to remove stop words automatically, so this step was unnecessary in preprocessing.

The model returned a numerical score ranging from -1 to 1, where -1 represents a very negative sentiment, while 1 represents a very positive sentiment. As such, once this score was computed, it was then multiplied by 10, and added to the overall rating of the movie the user provided feedback for. In queries which return a list of movies such as suggesting a specific movie based on genre, the order is determined by the overall rating of the movie. As such, the more feedback provided, the more new suggestions will be made for overall customization for the user.

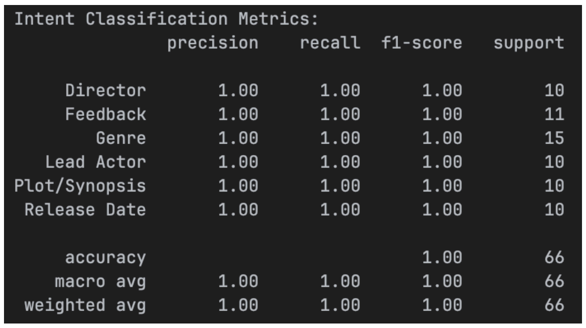
Data:

To effectively train the BERT model, a custom dataset was constructed featuring three key columns: the query, the intent, and the entity. Each query was crafted to train the model to adeptly respond to specific questions about critical movie aspects such as the director, lead actor, and release dates, as well as create recommendations based on another movie or a genre. This model is also trained to understand when feedback is being provided, and this is incorporated into the dataset. 100 unique queries were generated for each category to ensure comprehensive coverage and variety. This diversity allows the model to recognize and respond to a wide array of question formulations. Queries were formulated with questions like "Who was behind the direction of *The Hunger Games*?", "Who directed *Inception*?", and "Who was the creative mind directing *Divergent*?" This approach enables the model to understand and accurately answer different variations of similar questions, enhancing its usability in real-world applications where a user might inquire about movie details.

The IMDB movie dataset[3] was sourced from Kaggle and included essential information about movies such as genre, synopsis, title, director, and actors. The primary use of this dataset was to support the development of the recommendation system, providing a robust foundation for generating accurate and personalized movie suggestions.

Results:

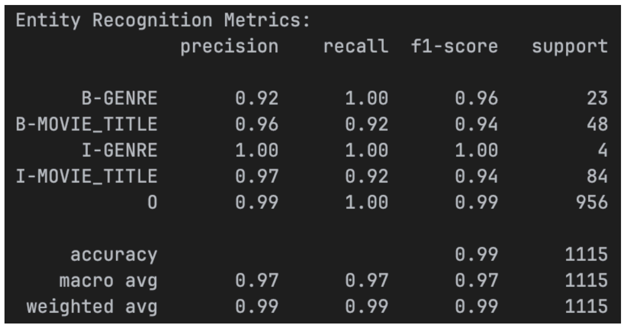
MovieMate has demonstrated notable success across various metrics of performance. Testing revealed that the DistilBert model, fine-tuned for specific requirements, achieved an impressive accuracy of 100% on the curated queries. However, when tested against phrases not included in the initial queries, the model, while still largely accurate, did not maintain perfect accuracy. These results underscore the model's effectiveness in understanding and categorizing user queries, which is crucial for delivering relevant movie information and recommendations.



```
Intent Classification Metrics:
              precision    recall  f1-score   support

     Director       1.00      1.00      1.00        10
     Feedback       1.00      1.00      1.00        11
        Genre       1.00      1.00      1.00        15
   Lead Actor       1.00      1.00      1.00        10
Plot/Synopsis       1.00      1.00      1.00        10
 Release Date       1.00      1.00      1.00        10

     accuracy                           1.00        66
    macro avg       1.00      1.00      1.00        66
 weighted avg       1.00      1.00      1.00        66
```



```
Entity Recognition Metrics:
               precision    recall  f1-score   support

     B-GENRE       0.92      1.00      0.96        23
B-MOVIE_TITLE       0.96      0.92      0.94        48
     I-GENRE       1.00      1.00      1.00         4
I-MOVIE_TITLE       0.97      0.92      0.94        84
           O       0.99      1.00      0.99       956

    accuracy                           0.99      1115
   macro avg       0.97      0.97      0.97      1115
 weighted avg       0.99      0.99      0.99      1115
```

This model also demonstrated efficiency in properly labeling named entities such as movie titles and genres, which are used to provide information or recommendations to the user.

The recommendation algorithms, leveraging TF-IDF scoring and cosine similarity, showed robust performance, albeit not flawless. This system was tested by generating movie recommendations across different genres. For instance, when analyzing movie similarity for "The Smurfs," a well-known children's movie noted for its animated characters, the model

[3] Dr. Shashikanth Vydyula. IMDB Movies - All Categories [Data set]

accurately recommended "Smurfs 2," the sequel, as the most relevant follow-up. Additional recommendations included other children's movies and cartoons, confirming that the model effectively recognized genre-specific characteristics.

The VADER sentiment analyzer was tested on the 100 example queries generated for the BERT model to correctly recognize feedback intent. Overall, the sentiment analyzer performed very well, classifying every positive query with a score greater than 0 and every negative query with a score less than 0.

Discussion:

The results demonstrate that MovieMate can significantly enhance user experience by providing personalized movie recommendations and accurate information retrieval. The high accuracy rate with the curated query set indicates that the DistilBert model is effectively categorizing user intents and recognizing named entities. However, the occasional slight decrease in accuracy when dealing with uncurated queries highlights the model's reliance on specific training data and its current scope of understanding. This suggests while the system performs well within the trained dataset's context, its adaptability to new, unexpected user inputs could be improved.

Limitations:

Despite its promising results, MovieMate has several limitations that could impact its effectiveness and user satisfaction:

- Typo Sensitivity: The current model's performance declines with typographical errors in entity names, which can be a common issue in user inputs. This limitation affects the system's ability to reliably interpret user queries and deliver accurate information.

- Intent Recognition Scope: With the system trained to recognize only six specific intents, its understanding is somewhat narrow, potentially overlooking user queries that fall outside these predefined categories.
- Content Depth: The movie dataset used has relatively brief synopsis, which may not provide enough context for generating nuanced recommendations that consider deeper thematic or narrative complexities.
- User Interface: Currently, MovieMate lacks a graphical user interface (GUI), which limits its accessibility and interactivity. A more user-friendly interface could significantly enhance user engagement and simplify interaction with the chatbot.

Future Work:

To address these limitations and enhance the overall functionality of MovieMate, the following future developments are proposed:

- Fuzzy Matching Implementation: Integrate fuzzy matching algorithms, such as Levenshtein distance, to better handle typos and variations in entity names, thus improving the robustness of the chatbot's responses.
- Expansion of Intent Recognition: By creating a more diverse set of training queries, the model could be trained to recognize a wider array of intents, making it more versatile and responsive to various user needs.
- Data Enrichment: Extend the movie database by scraping more detailed movie summaries from the web. This would provide a richer dataset for analysis, potentially improving the depth and relevance of the recommendations.

- GUI Development: Design and implement a graphical user interface that makes MovieMate more intuitive and accessible to users. A well-designed GUI would not only make the chatbot more appealing but also facilitate smoother interactions, making the system more engaging and easier to use.

By implementing these enhancements, MovieMate could further its lead in personalized entertainment solutions, offering more accurate, responsive, and user-friendly recommendations. These proposed developments are not only expected to address the current limitations but also to broaden the system's applicability and efficiency, ensuring a more engaging and satisfying user experience.

References:

Luoma, Jouni and Sampo Pyysalo. "Exploring Cross-sentence Contexts for Named Entity Recognition with BERT." *International Conference on Computational Linguistics* (2020).

Marcinczuk, Michal et al. "Text Document Clustering: Wordnet vs. TF-IDF vs. Word Embeddings." *Global WordNet Conference* (2021).

Hutto, C., and E. Gilbert. "VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text". *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 8, no. 1, May 2014, pp. 216-25, doi:10.1609/icwsm.v8i1.14550.

Chernev, A., Böckenholt, U., & Goodman, J. (2015). Choice overload: A conceptual review and meta-analysis. Journal of Consumer Psychology, 25(2), 333-358. https://doi.org/10.1016/j.jcps.2014.08.002

Dr. Shashikanth Vydyula. (2023). IMDB Movies - All Categories [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DSV/4941044

Appendix:

preprocess.py → File that contains function to preprocess data

- Define intents, BIO tags, encoders, tokenizer, and a dataset of queries to preprocess
- convert_to_bio → Take in a query, intent, and entity and returns a series that contains the intent, tokens, and BIO tags for the inputted query.
- encoder → Takes in a dataframe and encodes tokens, intents, BIO tags, and attention masks
- padding_func → takes in a batch and pads the input_ids, attention_masks, bio_labels, and intent labels in the batch. Assign padding value to -100 for BIO tags

data_loader.py → Class file to represent a dataset of movies.

- Used to efficiently pass information to the model.
- __init__ → initializes object with queries, attention_masks, intents, and bio_labels
- __len__ → gets the length of the dataset for testing and getting size
- __getitem__ → converts the queries, attention_masks, intents, and bio_labels to tensors

model_class.py → Class file that represents a model to perform intent and entity classification

- __init__ → Initializes object with number of entity and intent labels. Initialize the model as DistilBertForTokenClassification and add a linear layer to determine user intent. Use CrossEntropyLoss for both tasks.
- forward → forward pass of the model in which the loss is calculated by combining the loss from intent classification and entity recognition. Returns all loss values and logits, which are used to determine the predicted values.

class_training.py → File that contains model training steps

- Initialize a model filepath
- Create training, validation, and test sets and convert to MovieQueriesDataset
- train_model_class → Function to train a model using train and validation sets and save model to a file path

test_NER.py → File that contains test code for entity recognition and intent classification model

- Return the classification report for both intent classification and entity recognition on the test data.

querying_testing.py → File to run the chatbot using the trained model

- predict_intent_and_entities → find the intent and entities within a user's query using the trained model for entity recognition and intent classification
- chatbot_implementation → Implements chatting functionality to allow the user to communicate with the model in a conversational manner.

align_predictions.py → Contains function to align entity recognition labels with their tokens

- Align_predictions → Aligns outputs from model to original tokens to align entities with their BIO labels.

test_file.py → File to test all functions in the project

filterfunct.py → File that contains functions to filter a dataframe based on intent and entities.

- returnGenres → Returns a list of 10 movies of the given genre, sorted by rating.
- returnBasedOnMovie → Abstract function to return a passed in argument about a given movie. Is used by returnDirector, returnLeadActor, returnYear.
- returnDirector → Returns the director of a given movie.
- returnLeadActor → Returns the lead actor of the given movie.
- returnYear → Returns the year the movie was created.
- analyze_sentiment → Returns the sentiment of a given sentence from -1 to 1.
- change_sentiment → Changes the rating of the given movie, or the ratings of all movies of a given genre in the dataset based on a passed in sentiment score.

demo_moviemate.py → Contains a runnable demonstration of the chatbot using a list of queries

- simulate_chatbot_interaction → Function to demonstrate the functionality of the chatbot using an example list of queries. Each query gets passed into the input from the chatbot implementation function one at a time to mimic the interaction between the chatbot and a user.

old_model.py → Previously used file to train entity recognition and intent classification model

- Same preprocessing and preparation steps as class_training.py
- train_model → Function to train the model without a class implementation of the model. Model is created by attaching the linear layer directly to the imported DistilBert model. This implementation was discontinued due to a lack of ability to save the model parameters accurately and reload them into a model later.

filterfunct.py → File that contains functions to filter a dataframe based on intent and entities.

- preprocess_text → Preprocesses the given text by removing numbers, punctuation, stopwords, and lowercasing the text.
- get_similar_movies_index → Returns the indices of the most similar movies in the dataset from the given movie index.
- get_movie_index_by_title → returns the index of the given movie title in the dataset.
- main_get_similar_movies → returns the most similar movie titles to the given movie title.

User Manual:

Requirements: The required libraries for this project are located in the requirements.txt file
Unit tests for functions: runnable test_file.py
Demo file to test example queries: runnable demo_moviemate.py
**Run full chatbot interaction with custom queries: runnable querying_testing.py

If the model parameters file (NER_and_intent_class_model.pth) must be recreated, the model can be retrained by simply running the main function in the class_training.py file.

Note: Some of the test files do not run well with an anaconda interpreter so it is recommended to use the python 3.9 or 3.10 interpreter.