Islamic University of Technology (IUT)

Department of Computer Science and Engineering (CSE)

Predicting Tags for Movies from Plot Synopses Using Deep Learning Techniques

Authors

Md. Mezbaur Rahman - 154401

&

Saadman Malik - 154420

Supervisor

Raihan Islam Arnob

Lecturer, Department of CSE

A thesis submitted to the Department of CSE

in partial fulfillment of the requirements for the degree of B.Sc.

Engineering in CSE

Academic Year: 2018-2019

November - 2019

Declaration of Authorship

This is to certify that the work presented in this thesis is the outcome of the analysis and experiments carried out by Md. Mezbaur Rahman and Saadman Malik under the supervision of Raihan Islam Arnob, Lecturer of Department of Computer Science and Engineering (CSE), Islamic University of Technology (IUT), Dhaka, Bangladesh. It is also declared that neither of this thesis nor any part of this thesis has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Authors:

Md. Mezbaur Rahman

Student ID - 154401

Saadman Malik

Student ID - 154420

Supervisor:

Raihan Islam Arnob

Lecturer

Department of Computer Science and Engineering

Islamic University of Technology (IUT)

Acknowledgement

We would like to express our grateful appreciation for **Raihan Islam Arnob**, Lecturer, Department of Computer Science & Engineering, IUT for being our adviser and mentor. His motivation, suggestions and insights for this research have been invaluable. Without his support and proper guidance this research would never have been possible. His valuable opinion, time and input provided throughout the thesis work, from first phase of thesis topics introduction, subject selection, proposing algorithm, modification till the project implementation and finalization which helped us to do our thesis work in proper way. We are really grateful to him.

Abstract

Automatically generating or predicting tags for movies can help recommendation engines improve retrieval of similar movies, and help viewers know what to expect from a movie in advance. It improves the search results of a movie recommender system by predicting high weighted tags from a movie's plot synopsis. We propose a model in which we at first perform pre-processing of data(stopwords eradication, stemming of data etc.) and then tokenize the data by a technique called BERT and then vectorize it by TF-IDF process and then input those pre-processed data to a deep learning technique to give us a prediction tag scores from a set of tags for movies. We compare our system's result with an already proposed model with emotion flow encoded neural network and found that our model's performance shows improvement in result(TL, TR and F1 measure) specially due to pre-processing of data and for using the techniques like BERT and TF-IDF.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

A **recommender system** or a recommendation system (sometimes replacing "system" with a synonym such as platform or engine) is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. Recommendation systems use a number of different technologies. We can classify these systems into two broad groups[3],

- Content-based systems examine properties of the items recommended. For instance, if a Netflix user has watched many cowboy movies, then recommend a movie classified in the database as having the "cowboy" genre.

- Collaborative filtering systems recommend items based on similarity measures between users and/or items. The items recommended to a user are those preferred by similar users.

Tags for movies often represent summarized characteristics of the movies such as emotional experiences, events, genre, character types, and psychological impacts. As a consequence, tags for movies became remarkably convenient for recommending movies to potential viewers based on their personal preferences and user profiles. Tags can be used as strong search keywords and its an efficient feature for recommendation engines. The capability of tags in providing a quick glimpse of items can assist users to pick items precisely based on their taste and mood.

Natural language processing (NLP) is a field concerned about how to program computers to process and analyze large amounts of natural language data. Using NLP we can have a deep understanding of data and we can also take actions based on the outcomes of data. Natural language processing mainly deals with syntax, semantics, discourse and speech. Eradication of stopwords, lemmatization, stemming of words, removal of special characters, word segmentation etc. are all parts of natural language processing. It has a wide range of applications like Sentiment analysis, Text Classification, and Categorization, Automated Summarization etc.

Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on artificial neural networks. It can be supervised, semi-supervised and unsupervised. Artificial neural networks (ANN) are computing systems that are inspired by, but not identical to, biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules. Neural networks are workhorses for deep learning. Neural networks are multi-layer networks of neurons that we use to classify things, make predictions, etc. Neural network mainly consist of three layers which are input layer, zero,one or more than one hidden layers, output layer. These layers constitutes the neurons with activation function and these neurons are all connected together in a mesh network. ANN can be of many types like feed forward neural network, convolutional neural network (CNN), recurrent neural network (RNN) etc.

1.2 Problem Statement

The primary goal of a movie recommender system is to recommend movies to users based on their interests. This recommender system performs more efficiently when good tags are generated for each movies in its database. So when an user has a particular interest for a movie the recommender system finds the movies similar to the user's interest more quickly and accurately based on the generated tags for each movies. That means the more accurate the tags are generated for each movies, the better will be the result of the recommender system. The challenge for improving accuracy for better recommendation result is a burning research area at the moment. Recommender systems are used in many places like Netflix, Facebook, Google, YouTube etc. So our approach was to try to make a model from existing resources in order to improve the accuracy for generating tags from movie's plot synopses which will help improve a movie recommender system.

1.3 Motivation & Scopes

Recommender system is being used by many users in different platforms like Facebook, YouTube, Netflix etc. The better the accuracy of recommendation the more user friendly it will get. So in our thesis we mainly focused on movie recommender system by improving the scores for tag generation from a movie's plot synopses. An already implemented approach to this is done by using emotion flow encoded neural network[1]. After studying their methods then we focused on improving their results of tags learned(TL), tags recall(TR) and F1 measure by trying out a different model approach. On the mission to improve the accuracy we learned about data pre-processing which is a vital step in data mining related works, then we studied the techniques required for tokenizing and vectorizing and lastly to train the data by a deep learning technique to generate best scores from a set of given tags for movies.

The scope to improve the accuracy of our work will always remain open. Different techniques are being researched by different researchers and its being implemented to make new steps in a training model every now and then.

1.4 Research Challenges

The research challenges mainly includes:

- The time required to train the model. Some of our approach process required even days to complete all the epochs for training the data.
- High computational power with good specification PC is required for training the model.
- We tried two types of deep learning techniques in our model and when we find an improved result in tags learned(TL) then we find a bit lower result in F1 measure and vice-versa.

1.5 Thesis Outline

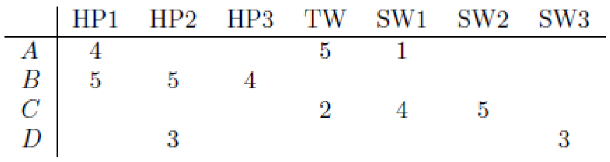
In Chapter 1 we have discussed our study in a precise and concise manner. Chapter 2 deals with the necessary literature review for our study and there development so far. In Chapter 3 we have stated the skeleton of our proposed method, proposed algorithm and also the flowchart to provide a detail insight of the working procedure of our proposed method. Chapter 4 shows the results and comparative analysis of successful implementation of our proposed method. The final segment of this study contains all the references and credits used.

2 Literature Review

2.1 Recommender System

Recommender system is such a facility which is an extensive class of web applications that involve predicting user responses to options. Its an information filtering system that predicts the preference an user would give to an item. We can introduce a model for recommender system based on utility matrix of preferences.

In a recommendation-system application there are two classes of entities, which we shall refer to as users and items. Users have preferences for certain items, and these preferences must be teased out of the data. The data itself is represented as a utility matrix, giving for each user-item pair, a value that represents what is known about the degree of preference of that user for that item. Values come from an ordered set, e.g., integers 1–5 representing the number of stars that the user gave as a rating for that item. We assume that the matrix is sparse, meaning that most entries are "unknown." An unknown rating implies that we have no explicit information about the user's preference for the item. [3]



	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Figure 1: Sample Utility Matrix[3]

Above figure representing a sample utility matrix with ratings of movies on a scale of 1 to 5. Notice that most user-movie pairs have blanks, meaning the user has not rated the movie. In practice, the matrix would be even sparser, with the typical user rating only a tiny fraction of all available movies. The goal of a recommendation system is to predict the blanks in the utility matrix. Based on the score of an user in a particular category of movie.
Considering the above figure would user A like SW2? There is little evidence

from the tiny matrix. We might design our recommendation system to take into account properties of movies, such as their producer, director, stars, or even the similarity of their names. If so, we might then note the similarity between SW1 and SW2, and then conclude that since A did not like SW1, they were unlikely to enjoy SW2 either. Alternatively, with much more data, we might observe that the people who rated both SW1 and SW2 tended to give them similar ratings. Thus, we could conclude that user A would also give SW2 a low rating, similar to user A's rating of SW1.[3]

Recommendation systems use a number of different technologies. We can classify these systems into two broad groups[3],

- Content-based systems examine properties of the items recommended. For instance, if a Netflix user has watched many cowboy movies, then recommend a movie classified in the database as having the "cowboy" genre. Content-Based systems focus on properties of items. Similarity of items is determined by measuring the similarity in their properties. In a content-based system, we must construct for each item a profile, which is a record or collection of records representing important characteristics of that item. Based on this characteristics of item the similarity measure with the item and user's interest is calculated by different processes like TF-IDF, Jaccard distance, Cosine distance etc. Then according to this measured value the high weighted item is recommended to user.[3]

- Collaborative filtering systems recommend items based on similarity measures between users and/or items. The items recommended to a user are those preferred by similar users. Collaborative-Filtering systems focus on the relationship between users and items. Similarity of items is determined by the similarity of the ratings of those items by the users who have rated both items. This similarity measure can be done by Jaccard distance or Cosine distance measurement.[3]

This diagram can give a proper understanding to these two categories of recommender system.

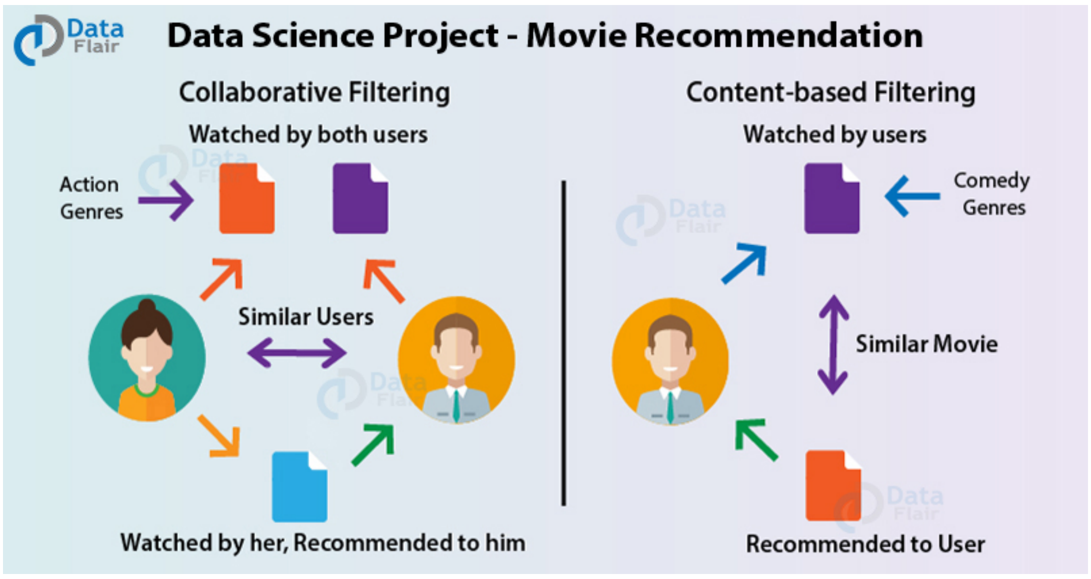


Figure 2: Recommender System Types[6]

2.2 Recommendation Goals

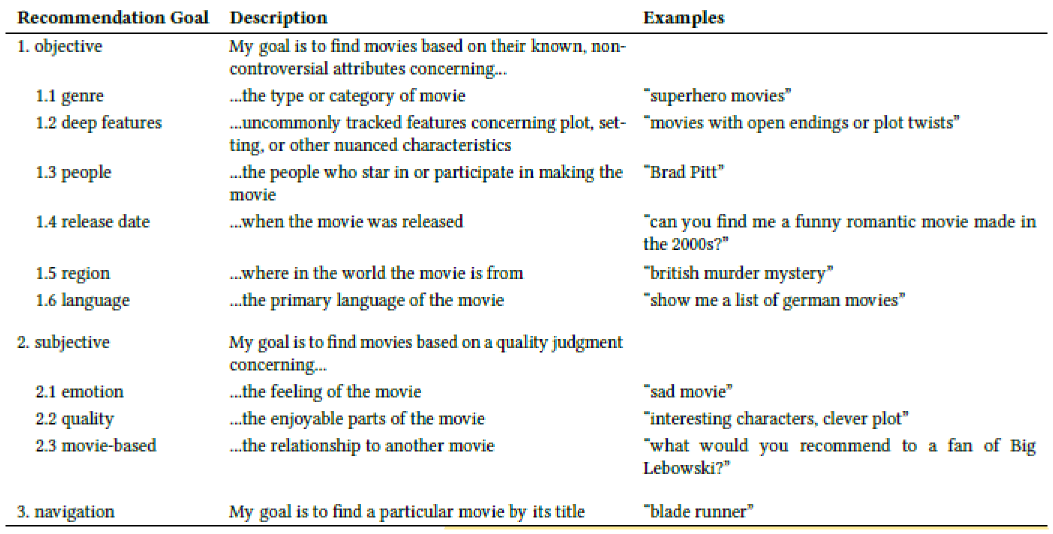
In the paper they developed three top level user recommendation goals which are described briefly below[4],

- Objective Goal: It's a request that can be answered without controversy. These goals seek to filter the movie space by specifying an attribute such as a genre, an actor, or a release date. It can be easily answered. However, they found many examples of objective goals that cannot be easily answered using a typical database of movie information. They label these goals as seeking **"deep features"**, indicating that users wish to filter movies by nuanced or specific criteria. Some examples of requests including deep features are "apocalyptic special effects and "a movie about berlin wall"[4].

- Subjective Goal: It's a request that involves judgment, uncertainty, and/or personalization. Answering subjective queries — much like objective deep features which is difficult because neither metadata databases nor recommender systems may track how much “clever plot” or “sad” a movie has. Subjective goals can be divided into three sub-types which are[4]:

  - **Emotion request** tend to specify a particular feeling that a movie invokes in the viewer, e.g. “cheerful comedy”.
  - **Quality requests** are either explicit about wanting good/best movies (e.g., “Some good dystopic sci-fi would be nice.”), or specify the aspects of the movie that make it good (e.g., “classic sci-fi movies’).
  - **Movie-based requests** seek related movies, e.g., “something like Pulp Fiction”. They considered movie-based requests to be subjective rather than objective, because there is no objective and universally-held metric to determine the similarity between any two movies

- Navigation Goal: They stated the navigation goal to be the simplest among the three goals — the user wants navigation to see one or more particular movies, so they state part or all of a title. Some examples in our dataset are “the social network” (which matches one movie) and “Star Wars” (which matches a series)[4].



Recommendation Goal	Description	Examples
1. objective	My goal is to find movies based on their known, non-controversial attributes concerning...	
1.1 genre	...the type or category of movie	"superhero movies"
1.2 deep features	...uncommonly tracked features concerning plot, setting, or other nuanced characteristics	"movies with open endings or plot twists"
1.3 people	...the people who star in or participate in making the movie	"Brad Pitt"
1.4 release date	...when the movie was released	"can you find me a funny romantic movie made in the 2000s?"
1.5 region	...where in the world the movie is from	"british murder mystery"
1.6 language	...the primary language of the movie	"show me a list of german movies"
2. subjective	My goal is to find movies based on a quality judgment concerning...	
2.1 emotion	...the feeling of the movie	"sad movie"
2.2 quality	...the enjoyable parts of the movie	"interesting characters, clever plot"
2.3 movie-based	...the relationship to another movie	"what would you recommend to a fan of Big Lebowski?"
3. navigation	My goal is to find a particular movie by its title	"blade runner"

Figure 3: Hierarchy of Coded Recommendation Goals.[4]

A user query can have more than one goals together, for example: “Funny romantic movie made in the 2000s” codes as genre (“romantic”), quality (“funny”).

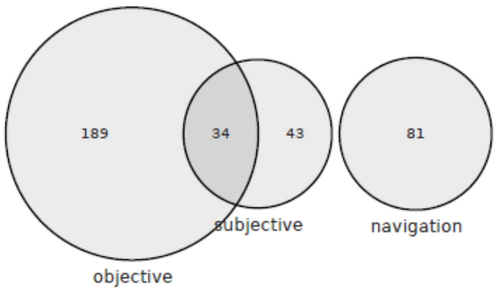


Figure 4: Venn Diagram of Top-Level Recommendation Goals.[4]

The other coding methods they used to describe user responses are described below briefly[4],

- Conversational: Some queries are phrased as though the user is conversing with a human; they code these queries as conversational. Examples include “I’m looking for a hard sci-fi movie” and “find a movie like eternal sunshine of the spotless mind”. They found that 24.8% of the queries in their dataset are conversational.

- Number of Modifiers: One measure of query complexity is the number of modifiers the query contains, where each modifier serves to filter or reorder the results. For example, “I’m looking for a movie that’s not sad” has a single modifier (“not sad”), while “biographic dramas” has two (“biographic”, “dramas”). In their dataset, 69.7% of the queries have a single modifier, 23.9% of the queries have two modifiers, 6.1% have three, and 0.3% have four (the maximum in their data).

- Recommend: Some queries in their dataset explicitly seek recommended movies. Some examples are “a good movie” and “I’m looking for the best sci fi horror movie". Only 4.4% of the queries in their dataset are explicit about seeking recommendations.

2.2.1 Follow-up Queries

Subjects who rate the results overall as "very poor" or "poor" are asked to explain how the results could be improved using free text input; subjects who rate the results as "fair" or better are asked to express a follow-up query (the interface prompts: "I can improve these results. Tell me more about what you want.").
Unlike subjects' first queries, where their goals are typically explicit and recognizable, follow-up queries are commonly ambiguous with respect to their goals. For instance, a subject whose first query is "Science Fiction" and whose follow-up query is "Horror" could plausibly either be specifying an additional genre filter, or could be starting a new search. So it becomes ambiguous.
The follow-up queries are divided into three parts which are briefly described below[4],

- <u>Refine:</u> The subjects assume the system remembers their initial query, and specify additional criteria that they wish the recommender system to consider. It can be divided into two sub-parts which are[4],
  - **Refine with further constraints:** Many refinement queries suggest that the subject is still interested in the initial query, and wishes to further constrain the universe of the search space.
  - **Refine with clarification:** Other refinement queries reflect a disappointment with the initial results. These subjects attempt to help the digital assistant by providing more information.
- <u>Reformulate:</u> These subjects appear to remain interested in their original query, but wish to completely restate the query to improve the recommendations. These subjects do not assume that the recommender remembers their last query, and typically reuse some portion of the original language. It can be divided into two sub-parts which are[4],

  - **Reformulate with further constraints:** As with the refine queries, some subjects appear to reformulate to further narrow the set of results.
  - **Reformulate with clarification:** Other subjects reformulate queries in an attempt to encourage the system to better results.

- <u>Start Over:</u> The third major theme they discovered in follow-up queries is that subjects want to start a new query, even though the experimental prompt says "tell me more about what you want" (emphasis not in the interface). These subjects may be experimenting with the system, or may realize that their first query is not at all what they are looking for.[4]

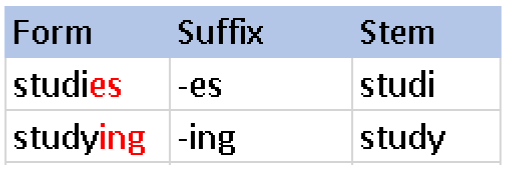
2.3 Natural Language Processing

By "natural language" we mean a language that is used for everyday communication by humans; languages such as English, Hindi, or Portuguese. In contrast to artificial languages such as programming languages and mathematical notations, natural languages have evolved as they pass from generation to generation, and are hard to pin down with explicit rules. We will take Natural Language Processing—or NLP for short—in a wide sense to cover any kind of computer manipulation of natural language. At one extreme, it could be as simple as counting word frequencies to compare different writing styles. At the other extreme, NLP involves "understanding" complete human utterances, at least to the extent of being able to give useful responses to them. Technologies based on NLP are becoming increasingly widespread. For example, phones and handheld computers support predictive text and handwriting recognition; web search engines give access to information locked up in unstructured text; machine translation allows us to retrieve texts written in Chinese and read them in Spanish. By providing more natural human-machine interfaces, and more sophisticated access to stored information, language processing has come to play a central role in the multilingual information society.[5] Natural language processing mainly deals with syntax, semantics, discourse and speech. Eradication of stopwords, lemmatization, stemming of words, tokenization, removal of special characters, word segmentation etc.

are all parts of natural language processing.

2.3.1 Stemming

It is possible to reduce certain words to a single word. Stemming is the concept of removing last characters of word until we get a common word. Or in other words, the stemming algorithm works by cutting out common suffixes from a word. A word stem doesn't have to be the same root as a morphological root based on a dictionary, it's just an equivalent or lower form of the word. Stemming algorithms are typically rule-based. A word is analyzed and passed through a set of conditionals that decide how it can be sliced. For example, we may have a suffix rule that, based on a list of known suffixes, cuts them off. In the English language, we have suffixes like "-ed" and "-ing" which may be useful to cut off in order to map the words "cook," "cooking," and "cooked" all to the same stem of "cook."



Form	Suffix	Stem
studies	-es	studi
studying	-ing	study

Figure 5: Example of Stemming

2.3.1.1 Overstemming and Understemming

Nevertheless, it is far from ideal since stemming is usually based on heuristics. Specifically, it is usually influenced by two issues: overstemming and understemming. Overstemming comes from being cut off too much of a term. This may result in ludicrous stems where all of the word's meaning is lost or muddled. Or it can lead to words being settled on the same roots, even though they should ideally not be. Take the four words university, universal, universities, and universe. A stemming algorithm that resolves these four words to the stem "univers" has overstemmed. While it might be nice to have universal and universe stemmed together and university and universities stemmed together, all four do not fit. A fair approach could have the first two resolved to "universe," and the latter two resolved to "universe," but the enforcement of rules could lead to more problems.

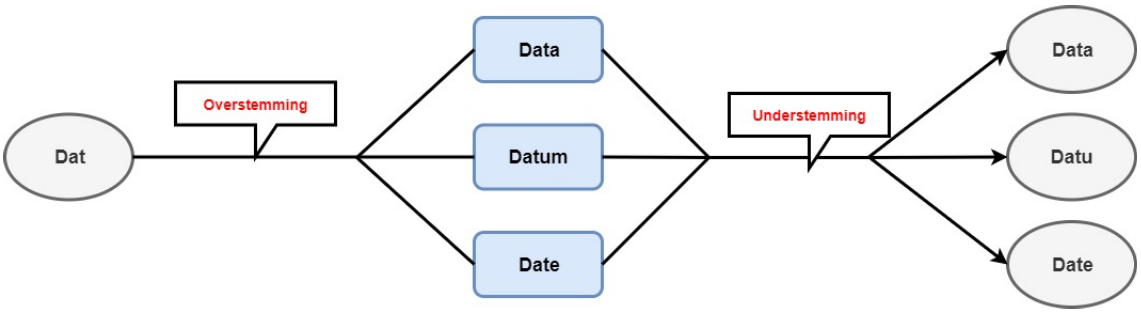


Figure 6: Overstemming and Understemming

Understemming is the other way around. This comes from having many words that are also each other's type. It would be good for them all to converge on the same base, but they don't, sadly. This can be seen if we have a stemming algorithm that stems the words data and datum to "dat" and "datu." What happens if we resolve both of them to "dat". What are we going to do with date, though? And there's a good rule in general? Or are we just for a very specific example implementing a very specific rule? There are a number of stemming algorithms but the most common algorithms are Porter, Lancaster and Snowball.

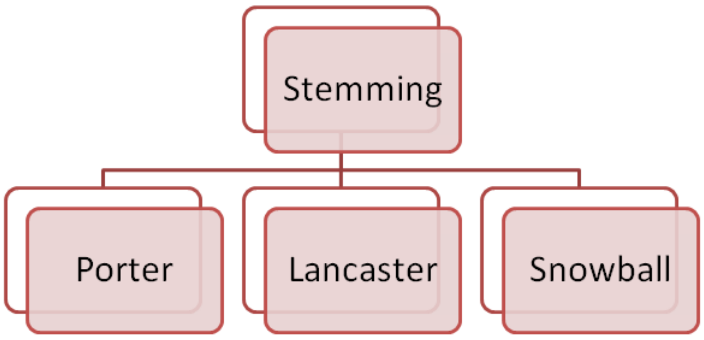
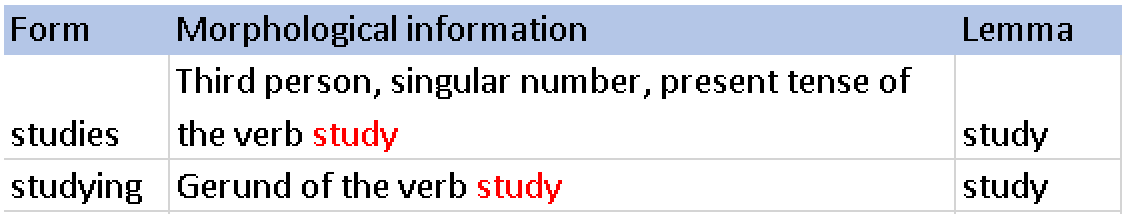


Figure 7: Types of Stemming

2.3.2 Lemmatization

On the other hand, lemmatization takes into account the morphological analysis of the terms. To do this, it is important to have comprehensive dictionaries that can be looked through by the algorithm to connect the form back to its lemma. Once, with the same example of words, you can see how it works.



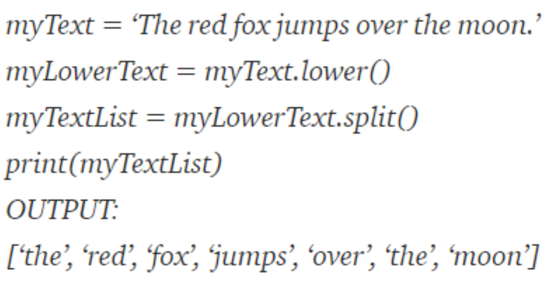
Form	Morphological information	Lemma
studies	Third person, singular number, present tense of the verb study	study
studying	Gerund of the verb study	study

Figure 8: Lemmatization

To solve a word to her lemma, lemmatization needs to know its part of the speech. This requires additional computational linguistic power such as a talking tagger component. This makes it possible to make better resolutions(like resolving is and are to “be”).

2.3.3 Tokenization

We will start with some very simple text parsing to get started in natural language processing. Tokenization is the method of taking a text stream like a sentence and breaking it down to its most basic words. For instance take the following sentence: “The red fox jumps over the moon.” Each word would represent a token of which there are seven. To Tokenize a sentence using python:



```
myText = ‘The red fox jumps over the moon.’
myLowerText = myText.lower()
myTextList = myLowerText.split()
print(myTextList)
OUTPUT:
[‘the’, ‘red’, ‘fox’, ‘jumps’, ‘over’, ‘the’, ‘moon’]
```

Figure 9: Tokenization

2.3.4 Stop Word Removal

Most sentences and paragraphs contain terms whose meaning or significance is very little. These words include “a,” “and,” “an,” and “the.” Stop word removal is a process of removing these words from a sentence or stream of words.

2.4 Deep Learning

Deep learning is a method of artificial intelligence that mimics the functioning of the human brain in processing data and generating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning from unstructured and unlabeled information without supervision. Also known as deep neural learning or deep neural network. Deep learning learns from vast amounts of unstructured data that could normally take humans decades to understand and process.

2.4.1 Feed Forward Neural Network(FFNN)

Neural networks are multi-layer networks of neurons (the blue and magenta nodes in the chart below) that we use to classify things, make predictions, etc. Below is the diagram of a simple neural network with five inputs, 5 outputs, and two hidden layers of neurons.[7]

Starting from the left we have in Figure:9

1. The input layer of our model in orange.
2. Our first hidden layer of neurons in blue.
3. Our second hidden layer of neurons in magenta.
4. The output layer (a.k.a. the prediction) of our model in green.

The arrows that connect the dots shows how all the neurons are interconnected and how data travels from the input layer all the way through to the output layer. So in the neural network,

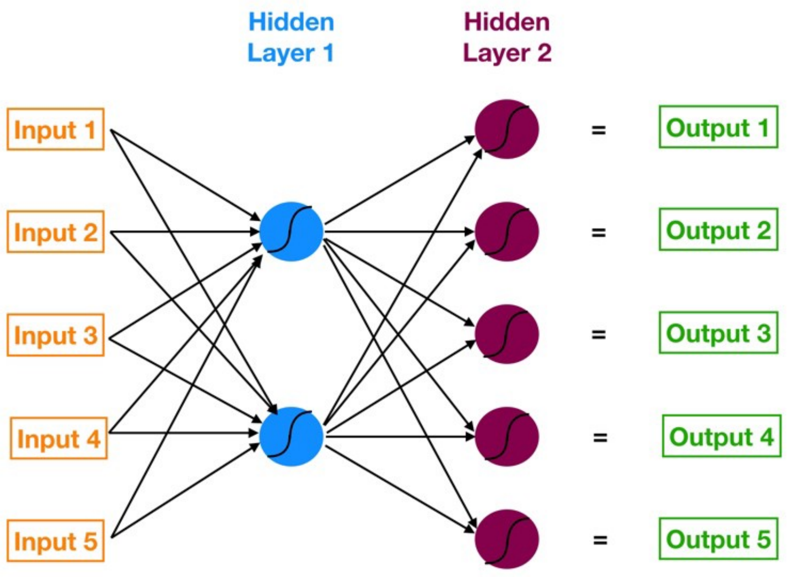


Figure 10: Feed Forward Neural Network[7]

1. The input(orange) is the lone feature that we give to our model in order to calculate a prediction.

2. The blue or magenta neurons are the hidden layers consisting of activation function which takes product of weight and input value as the neurons input and output it to another hidden layer or to the output layer according to the activation function like sigmoid function.

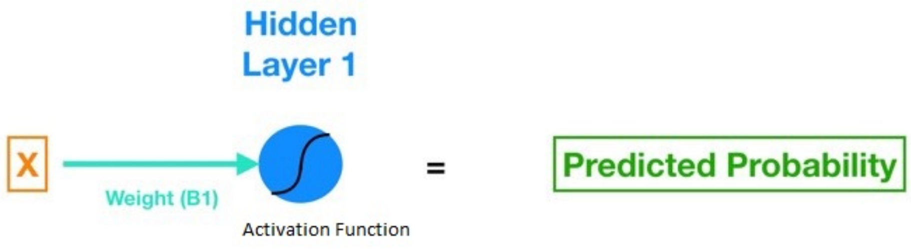


Figure 11: Activation Function in Neural Network[7]

3. Finally we get our predicted probability score as our output(green) from the neurons depending on the activation function.

2.4.2 Encoding Emotion Flow with a Neural Network

This neural network model was showed in a paper in which they explore the problems of creating tags for movies from their plot synopses and with that they propose a novel neural network model that merges information from synopses and emotion flows throughout the plots to predict a set of tags for movies.[1] They conducted the experiment on the Movie Plot Synopses with Tags (MPST) corpus which is a collection of plot synopses for 14,128 movies collected from IMDB and Wikipedia.[11]
The emotion flow encoded neural network model is described below:

1. Convolutional Neural Network (CNN): At first from the plot synopses they design a model that takes word sequences as input, where each word is represented by a 300-dimensional word embedding vector. They stacked 4 sets of one-dimensional convolution modules with 1024 filters each for filter sizes 2, 3, 4, and 5 to extract word-level n-gram features. Convolution units of filter size c calculate a convolution output using a weight map Wc, bias bc, and the ReLU activation function. Then Maxpool operation is applied to these convolution outputs and have taken the maximum value as the feature produced a particular filter. From these we get out textual features of the plot synopses.[1]

2. Bidirectional Long Short Term Memory (Bi-LSTM): To model the flow of emotions throughout the plots, we divide each synopsis into N equally-sized segments based on words. For each segment, we compute the percentage of words corresponding to each emotion and polarity type (positive and negative) using the NRC emotion lexicons. NRC emotion lexicons is a list of 14,182 words and their binary associations with eight types of elementary emotions from the Hourglass of Emotions model (anger, anticipation, joy, trust, disgust, sadness, surprise, and fear) with polarity. These lexicons have been used effectively in tracking the emotions in literary texts and predicting success of books.

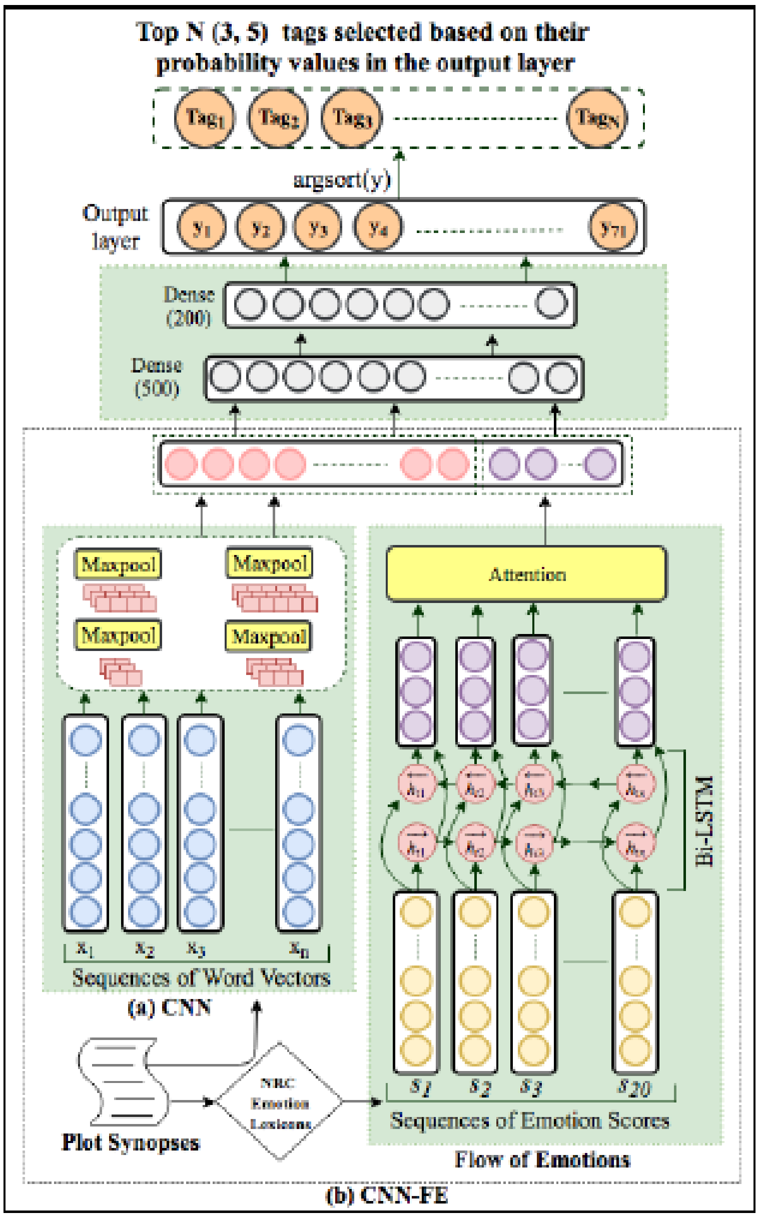


Figure 12: Emotion Flow Encoded Neural Network[1]

This bidirectional LSTM layer tries to summarize the contextual flow of emotions from both directions of the plots. The forward LSTMs read the sequence from s1 to sN, while the backward LSTMs read the sequence in reverse from sN to s1. These operations will compute the forward hidden states and backward hidden states. For input sequence s, the hidden states ht are computed using the following intermediate calculations:

$$\begin{aligned} i_t &= \sigma(W_{si^{st}} + W_{hi}h_{t-1} + W_{ci^{c_{t-1}}} + b_i) \\ f_t &= \sigma(W_{sf^{st}} + W_{hf}h_{t-1} + W_{cf^{c_{t-1}}} + b_f) \\ c_t &= f_{t^{c_{t-1}}} + i_t tanh(W_{sc^{st}} + W_{hc^{h_{t-1}}} + b_c) \\ o_t &= \sigma(W_{sc^{st}} + W_{hc}h_{t-1} + b_c) \\ h_t &= o_t tanh(c_t) \end{aligned}$$

where, W and b denote the weight matrices and bias, respectively. σ is the sigmoid activation function, and i, f, o, and c are input gate, forget gate, output gate, and cell activation vectors, respectively. Then they apply attention mechanism on this representation to get a unified representation of the emotion flow.[1]

3. Feed Forward Neural Network with two hidden layers: From CNN we get textual feature and from Bi-LSTM we get emotional features of the plot synopses and they concatenate the representation of the emotion flow produced by the attention operation and the output vector with the vector representation generated from the CNN module. The concatenated vector is then fed into two hidden dense layers with 500 and 200 neurons. To improve generalization of the model, we use dropout with a rate of 0.4 after each hidden layer. Finally, we add the output layer with 71 neurons to compute predictions for 71 tags. Then they normalize the output layer by applying a softmax function which gives us the probabilities of those 71 tags in the output layer. Then they chose the tags with high probability value.[1]

2.4.3 Long Short Term Memory(LSTM)

Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory. So before we describe it further we say a bit about recurrent neural network.
Recurrent Neural Network is a generalization of feedforward neural network that has an internal memory. RNN is recurrent in nature as it performs the same function for every input of data while the output of the current input depends on the past one computation. After producing the output, it is copied and sent back into the recurrent network. For making a decision, it considers the current input and the output that it has learned from the previous input.[9]

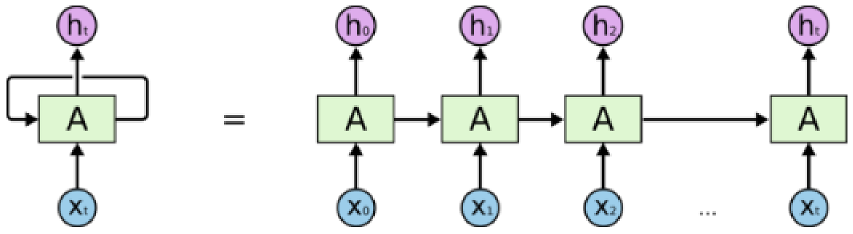


Figure 13: Recurrent Neural Network[9]

The vanishing gradient problem of RNN is resolved by LSTM. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation.[9] The LSTM network has three gates:

1. **Input Gate:** It discovers which value from input should be used to modify the memory. Sigmoid function decides which values to let through 0,1 and tanh function gives weightage to the values which are passed deciding their level of importance ranging from -1 to 1.[9]

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$$
$$C_t = tanh(W_c.[h_{t-1}, x_t] + b_c)$$

2. **Forget Gate:** It discovers what details to be discarded from the block. It is decided by the sigmoid function. It looks at the previous state(ht-1) and the content input(Xt) and outputs a number between 0(omit this)and 1(keep this)for each number in the cell state Ct1.[9]

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f)$$

3. **Output Gate:** The input and the memory of the block is used to decide the output. Sigmoid function decides which values to let through 0,1. and tanh function gives weightage to the values which are passed deciding their level of importance ranging from-1 to 1 and multiplied with output of Sigmoid.[9]

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * tanh(C_t)$$

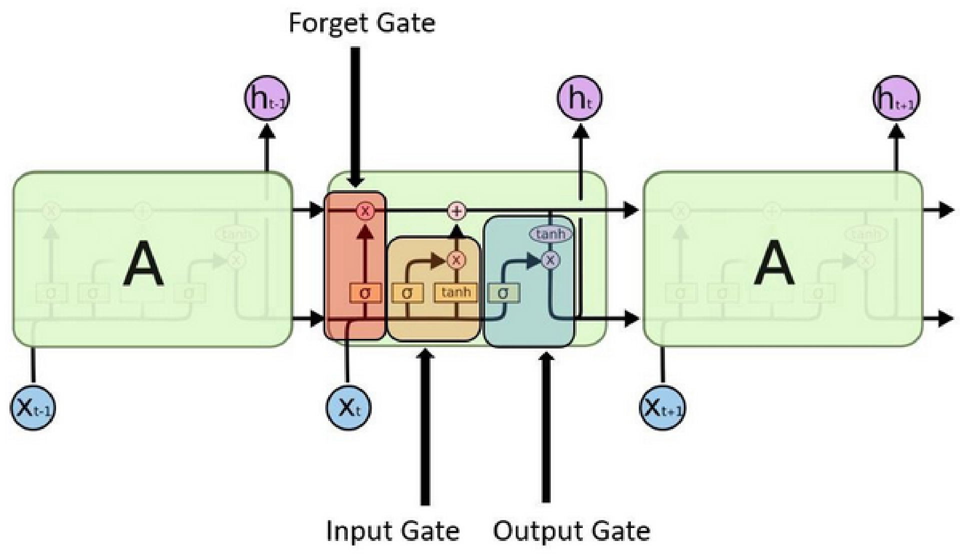


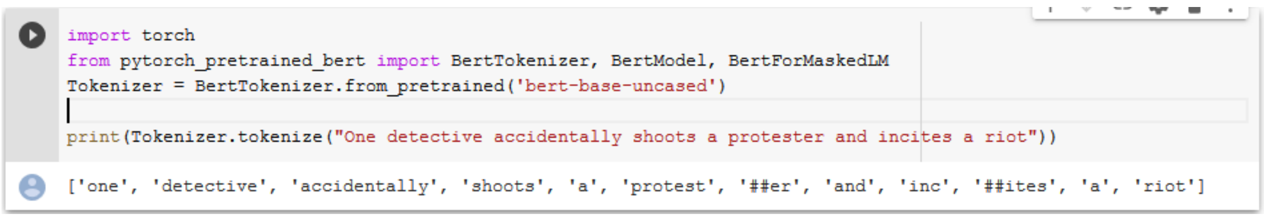
Figure 14: LSTM Gates[9]

2.5 BERT: Bidirectional Encoder Representations from Transformers

BERT, which stands for Bidirectional Encoder Representations from Transformers, is a neural network-based technique for natural language processing pre-training. For example, in the phrases "nine to five" and "a quarter to five," the word "to" has two different meanings, which may be obvious to humans but less so to search engines. BERT is designed to distinguish between such nuances to facilitate more relevant results. The breakthrough of BERT is in its ability to train language models based on the entire set of words in a sentence or query (bidirectional training) rather than the traditional way of training on the ordered sequence of words (left-to-right or combined left-to-right and right-to-left). BERT allows the language model to learn word context based on surrounding words rather than just the word that immediately precedes or follows it.[2]

2.5.1 BERT Tokenization

Anyone, of course, will intrigued by the idea of a single model with a broad common vocabulary for 104 languages. This is exactly what BERT(Bi-directional Representation from Transformers) offers. The vocabulary is 119,547 WordPiece model, and the input is tokenized into word pieces (also known as subwords) so that each word piece is an element of the dictionary. Non-word-initial units are prefixed with '' as a continuation symbol except for Chinese characters which are surrounded by spaces before any tokenization takes place. The tokenizer favors longer word pieces with a de facto character-level model as a fallback as every character is part of the vocabulary as a possible word piece.[8] An example of such tokenization using PyTorch implementation of BERT looks like this:



```
import torch
from pytorch_pretrained_bert import BertTokenizer, BertModel, BertForMaskedLM
Tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

print(Tokenizer.tokenize("One detective accidentally shoots a protester and incites a riot"))
```

```
['one', 'detective', 'accidentally', 'shoots', 'a', 'protest', '##er', 'and', 'inc', '##ites', 'a', 'riot']
```

Figure 15: PyTorch implementation of BERT

This segmentation is purely frequency based and it is very different from what a true morphological segmenter would output (El-végez-het-itek). This example is certainly longer than an average Hungarian word and the average word is not tokenized this aggressively but BERT does produce a large number of word pieces for certain languages.[8]

2.6 Bag of Words and Term Frequency-Inverse Document Frequency

A common technique for extracting features from text in natural language processing is to put all the words that appear in the text in a bucket. This aproach is called a bag of words model or BoW for short. It's referred to as a "bag" of words because any information about the structure of the sentence is lost.

Bag of Words (BoW) is an algorithm that counts how many times a word appears in a document. It's a tally. Such counts of words allow us to compare documents and determine their similarities for applications such as search, identification of documents and topics modeling. BoW is a also method for preparing text for input in a deep-learning net. BoW lists terms combined with each document's word counts. Each row is a word, each column is a document, and each cell is a word count in the table where the words and documents that actually become vectors are stored. Each of the documents in the corpus is represented by columns of equal length. Those are wordcount vectors, an output stripped of context.

Words	Doc1	Doc2	Doc3
car	27	4	24
auto	3	33	0
insurance	0	33	29
best	14	0	17

Table 1: Example of Bag of Words (BoW)

In the table considering the first row it means that the word 'car' is found 27 times in Doc1, 4 times in Doc2, 24 times in Doc3 and the rest of the rows are created similarly.

Term-frequency-inverse document frequency (TF-IDF) is another way of judging an article's subject by its words. With TF-IDF, words are given weight – TF-IDF measures relevance, not frequency. That is, word counts are replaced throughout the entire dataset with TF-IDF scores. First, TF-IDF calculates the number of times that words appear in a particular document (i.e. "term frequency"). But because words like "and" and "the" often appear in all documents they need to be discounted systematically. That's the inverse-document frequency part. The more documents a word appears in, the less valuable that word is as a signal to differentiate any given document. That's intended to leave only the frequent AND distinctive words as markers. Each word's TF-IDF relevance is a normalized data format that also adds up to one.

$$W_{i,j} = tf_{i,j} * log(\frac{N}{df_i})$$

$$tf_{i,j} = \text{number of occurences of i in j}$$

$$df_i = \text{number of document containing in i}$$

$$\text{N} = \text{total number of documents}$$

Those marker words are then fed to the neural net as features in order to determine the topic covered by the document that contains them. The main difference is that Word2vec produces one vector per word, whereas BoW produces one number (a wordcount). Word2vec is great for digging into documents and identifying content and subsets of content. Its vectors represent each word's context, the ngrams of which it is a part. BoW is a good, simple method for classifying documents as a whole.

3 Proposed Method

3.1 Skeleton of Proposed Method

Our method predict tags from movie's plot synopses using deep learning techniques. At first we do pre-processing of data in our dataset. Pre-processing of data includes removal of stopwords, removal of special characters, stemming of data, lemmatization. Then we tokenize the input data into sub pieces by BERT tokenizer. Afterwards we use TF-IDF vectorizer to generate a score of importance of a particular tokens for a particular movie's plot synopses. Thus our X(train) data is formed. Our Y(train) data is created by count vectorizer which keeps the count of the tags occured for a particular movie's plot synopses in training data. With our X(train) and Y(train) data we input it in our neural network to generate a model. Then we input our test data in that model to generate probabilites of a total of 71 tags and the those tags with highest probability score (Top 1, Top 3, Top 5, Top 10) is selected to be the tags for that particular movie's plot synopses. We also made another model by using LSTM with different constraints where we got another set of results for different movies plots.

3.2 Proposed Algorithm

Our proposed algorithm describing all the steps are given below:

3.2.1 Dataset

We conduct our experiments on the Movie Plot Synopses with Tags (MPST) corpus, which is a collection of plot synopses for 14,828 movies collected from IMDb and Wikipedia. Most importantly, the corpus provides one or more fine-grained tags for each movie. The reason behind selecting this particular dataset is two-fold. First, the tagset is comprised of manually curated tags. These tags express only plot-related attributes of movies (e.g. suspenseful, violence, and melodrama) and are free of any tags foreign to the plots, such as metadata. Furthermore, grouping semantically similar tags and representing them by generalized tags helped to reduce the noise created by redundancy in tag space. Second, the corpus provides adequate amount of texts in the plot synopses as all the synopses have at least ten sentences. We follow the same split provided with the corpus, using 80% for training and 20% for test set.[1, 11] The statistics of the dataset is given below:

Split	Plot Synopses	Tags	Tags per Movie	Sentence per Synopsis	Words per Synopsis
Train	11862	71	2.97	42.36	893.39
Test	2966	71	3.04	42.61	907.96

Table 2: Statistics of MPST corpus[1]

3.2.2 Pre-Processing of Data

Pre-processing of data is done by Natural Language Toolkit (NLTK) in python. Data pre-processing steps are given below:

1. **Removal of stopwords:** Stopwords such as 'a', 'an', 'the' etc which are very frequently used in many sentences with no significant purpose are removed for better processing of data. We implemented it using the following code,[10]

```python
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
stopwords = set(stopwords.words('english'))
```

2. **Removal of special characters:** Special characters or punctuations like ',', '.', '' are removed to help processing of data. We implemented it in our code by,[10]

```python
import re
sentence = re.sub('[^A-Za-z]+', ' ', sentence)
```

3. **Stemming and Lemmatization:** Stemming and lemmatization has been elaborately discussed in chapter 02. Among the three types of stemmer we

used snowball stemmer. We implemented stemming and lemmatization of data by the following python code:[10]

```
import nltk
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
stemmed_sentence = []
for e in sentance.split():
    if e.lower() not in stopwords:
        s=(sno.stem(lemmatizer.lemmatize(e.lower()))).
            encode('utf8')
        stemmed_sentence.append(s)
sentance = b'␣'.join(stemmed_sentence)
```

3.2.3 Tokenizing and Vectorizing

The data is tokenized using BERT tokenizer and these tokenized sub pieces of words of plot synopses are used in TF-IDF vectorizer. The supervised tags for each movie's plot synopsis in training data is vectorized using count vectorizer. Our implementation is given below:

- **Tokenization**

```
!pip install pytorch-pretrained-bert
import torch
from pytorch_pretrained_bert import BertTokenizer,
    BertModel, BertForMaskedLM

Tokenizer = BertTokenizer.from_pretrained('bert-base-
    uncased')
Tokenizer.tokenize("This␣here's␣an␣example␣of␣using␣
    the␣BERT␣tokenizer␣using␣tokenize␣this␣brother")
```

- **Count Vectorizer**[10]

```
vectorizer = CountVectorizer(tokenizer = lambda x: x.
    split(',␣'),binary='true')
y_train_vect = vectorizer.fit_transform(yTrain)
y_test_vect = vectorizer.transform(yTest)
```

- **TF-IDF Vectorizer**

```
vectorizer = TfidfVectorizer(min_df=10,max_features
    =20000,smooth_idf=True,norm="l2", tokenizer=lambda x
    :bert_tokenizer.tokenize(x),sublinear_tf=False,
    ngram_range=(1,4))
x_train_ngram = vectorizer.fit_transform(xTrain)
x_test_ngram = vectorizer.transform(xTest)
```

3.2.4 Deep Learning Techniques

We use two different deep learning techniques to generate two set of results. The techniques are given below:

3.2.4.1 Feed Forward Neural Network

Our implementation of Feed Forward Neural Network is given below:

```
import numpy as np
np.random.seed(1234)


import tensorflow as tf
tf.keras.optimizers.Adam(learning_rate=0.0001, beta_1=0.9,
    beta_2=0.999, amsgrad=False)


model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu)
    )
model.add(tf.keras.layers.Dropout(0.35))
model.add(tf.keras.layers.Dense(64, activation=tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.35))
model.add(tf.keras.layers.Dense(71, activation=tf.nn.
    sigmoid))
model.compile(optimizer='adam', loss='
    categorical_crossentropy', metrics=['accuracy'])

model.fit(X_new_train,Y_new_train,epochs=50)
pred = model.predict(X_new_test)
```

3.2.4.2 Long Short Term Memory(LSTM)

Our implementation of Long Short Term Memory(LSTM) is given below:

```
import numpy as np
np.random.seed(1234)

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM

model = Sequential()
model.add(LSTM(128, input_shape=(X_new_train.shape[1:]),
    activation='relu', return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(71, activation='sigmoid'))
```

```
opt = tf.keras.optimizers.Adam(lr=0.0001, decay=1e-6)
model.compile(
    loss='categorical_crossentropy',
    optimizer=opt,
    metrics=['accuracy'],
)
model.fit(X_new_train,Y_new_train,epochs=350)
pred = model.predict(X_new_test)
```

3.2.5 Outputs

The probability score of the tags for the movies plot synopses for test data is extracted in pickle file format of python similar to the following code:

```
with open(OUTPUT_PATH+'test_lstm5.pickle', 'wb') as f:
    pickle.dump(Y_new_test, f)
```

Then this output file is evaluated to generate the values of Tags Learned(TL), F1 measure and Tags Recall(TR). The code for generating these evaluation metrics are given below:

- **Output file extraction**

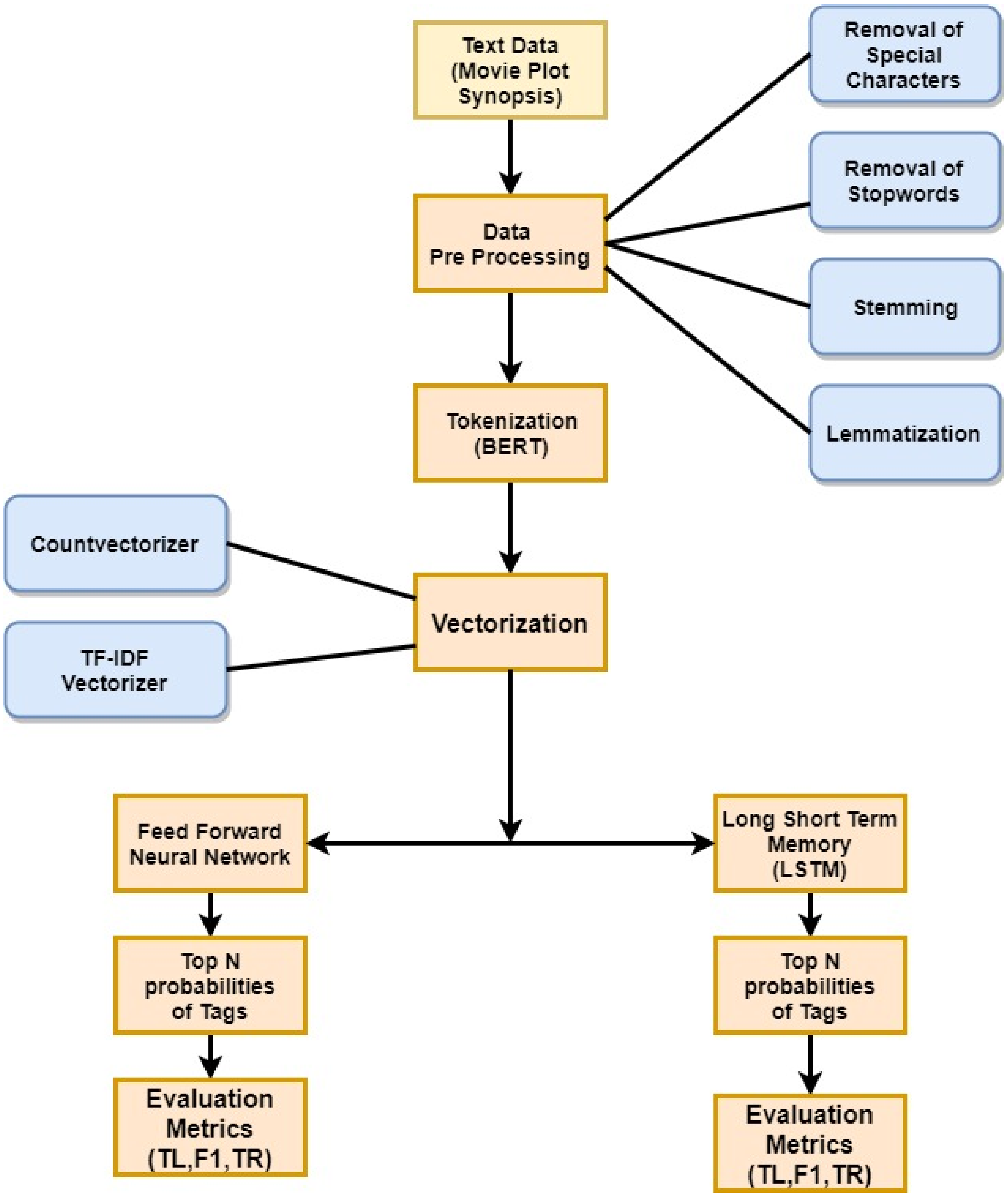
```
import pandas as pd
import numpy as np
from report import EvaluationReports
import pickle
top_n_list = [1, 3, 5, 8, 10]

OUTPUT_PATH ='../output/'
with open(OUTPUT_PATH+'pred_nn_gpt.pickle', 'rb') as f
    : predicted_probabilities_list = pickle.load(f)
with open(OUTPUT_PATH+'test.pickle', 'rb') as f:
    target_tags_list = pickle.load(f)
```

```
T =  np.array(target_tags_list).squeeze()
P=  np.array(predicted_probabilities_list).squeeze()
E = EvaluationReports()
results = E.get_tl_f1_and_tr(P,T,top_n_list)

print('now')
print(results)
```

- **Evaluation of Outputs**

```
from sklearn.metrics import f1_score
from sklearn.metrics import hamming_loss
from sklearn.metrics import accuracy_score
from sklearn.metrics import average_precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import zero_one_loss
from sklearn.metrics import
    label_ranking_average_precision_score
from sklearn.metrics import coverage_error
import numpy as np
import pandas as pd

class EvaluationReports:

    @staticmethod
    def get_top_n_predictions(pred_prob_matrix, top_n)
        :
        sorted_idx = np.argsort(pred_prob_matrix)
        for i in range(sorted_idx.shape[0]):
            sorted_idx[i] = sorted_idx[i][::-1]
        outputs = []
        unique_predictions = []

        for tn in top_n:
            sliced = sorted_idx[:, :tn]
            unique_set = set(sliced.flatten())
            one_hot = np.zeros(pred_prob_matrix.shape)
            for idx in range(sliced.shape[0]):
                one_hot[idx][sliced[idx]] = 1
            outputs.append(one_hot)
            unique_predictions.append(len(unique_set))

        return outputs, unique_predictions

    def get_tr(self,y_pred,y_true):
        pos=np.zeros((71,1))
        tpos=np.zeros((71,1))
        recall=0.0
        for j in range(y_pred.shape[0]):
            true=y_true[j]
            pred=y_pred[j]
            for i in range(pos.shape[0]):
                if(true[i]==1):
                    pos[i]=pos[i]+1
                if(true[i]==1 and pred[i]==1):
                    tpos[i]=tpos[i]+1

        for i in range(71):
            recall = recall + (tpos[i]/pos[i])
        tr = recall/71.0
        return tr[0]

    def get_tl_f1_and_tr(self, y_pred, y_true, top_n):
        y_pred = y_pred
        y_true = y_true
        top_preds, learned_tags = self.
           get_top_n_predictions(y_pred, top_n)
        output = []

        for tp, lt in zip(top_preds, learned_tags):
            micro_f1 = f1_score(y_true, tp, average='
               micro')
            tr= self.get_tr(tp,y_true)
            output.append([round(lt, 5),round(micro_f1
               , 5),round(tr,5)])
        return output
```

3.3 Flow Diagram



Text Data (Movie Plot Synopsis)
Data Pre Processing
Removal of Special Characters
Removal of Stopwords
Stemming
Lemmatization
Tokenization (BERT)
Countvectorizer
TF-IDF Vectorizer
Vectorization
Feed Forward Neural Network
Long Short Term Memory (LSTM)
Top N probabilities of Tags
Top N probabilities of Tags
Evaluation Metrics (TL,F1,TR)
Evaluation Metrics (TL,F1,TR)

Figure 16: Flow Diagram of our Proposed Method

4 Results & Discussion

4.1 Experimental Result

Our experimental results are evaluated based on three metrics which are:

- **Tags Learned (TL):** This value indicates the number of tags learned by the model.
- **Tags Recall (TR):** Tag recall value is calculated based on the following formula

  $$\frac{\sum_{i=1}^{|T|} R_i}{|T|}$$

  where $|T|$ is the total number of tags in the corpus and R_i is the recall for the i^{th} tag.
- **F1 measure:** The F1 measure is the harmonic mean, or weighted average, of the precision and recall scores. So the formula to calculate F1 measure is

  $$2 * \frac{Precision * Recall}{Precision + Recall}$$

The experimental result values are shown for two different models in two different tabular formats which is sectioned below:

4.1.1 With Feed Forward Neural Network

The result table for Feed Forward Neural Network is shown below:

Number of Epochs	Layers	Learning Rate	Unit Values in Layers	Top 01			Top 03			Top 05			Top 08			Top 10		
				TL	F1	TR	TL	F1	TR	TL	F1	TR	TL	F1	TR	TL	F1	TR
50	04	0.0001	(128,64,32,71) Dropout: 0.35	30	**24.68**	3.75	51	**35.57**	9.13	59	**35.31**	14.22	67	**32.06**	21.43	**71**	**29.64**	26.75
50	04	0.0001	(256,128,64,71) Dropout: 0.35	**63**	24.16	4.7	**68**	35.11	**11.76**	**69**	34.88	**17.76**	**71**	31.39	24.71	**71**	28.92	28.88
50	03	0.0001	(128,64,71) Dropout: 0.35	60	24.11	**4.8**	**68**	34.83	11.18	**69**	34.6	17.07	**71**	31.35	**25.43**	**71**	28.94	**31.19**

Table 3: Results with Feed Forward Neural Network

4.1.2 With Long Short Term Memory

The result table for Long Short Term Memory is shown below:

Number of Epochs	Layers	Learning Rate	Unit Values in Layers	Top 01			Top 03			Top 05			Top 08			Top 10		
				TL	F1	TR	TL	F1	TR	TL	F1	TR	TL	F1	TR	TL	F1	TR
50	04	0.001	(128 lstm,128 lstm, 32 dense,71 dense)	58	23.22	4.12	67	33.59	10.37	70	33.76	15.89	**71**	31.10	23.37	**71**	28.46	27.79
150	03	0.0001	(128 lstm,64 lstm, 71 dense)	56	**23.82**	**4.5**	65	**34.15**	11.11	69	**34.74**	**16.89**	**71**	**31.72**	**24.84**	**71**	**29.37**	**30.14**
200	03	0.0001	(128 lstm,64 lstm, 71 dense)	**63**	22.92	**4.5**	**68**	33.56	**11.16**	**71**	33.77	16.55	**71**	30.96	24.68	**71**	28.70	29.69

Table 4: Result with Long Short Term Memory(LSTM)

4.2 Comparative Analysis

Comparative results of our work with previous related works can be shown by the following table:

Methods	Top 03			Top 05			Top 10		
	TL	F1	TR	TL	F1	TR	TL	F1	TR
Baseline: Most Frequent	3	29.7	4.23	5	28.4	14.08	10	28.4	13.73
Baseline: Random	**71**	4.2	4.21	**71**	6.4	15.04	**71**	6.6	14.36
Baseline: Kar et al. (2018)	47	**37.3**	**10.52**	52	**37.3**	**16.77**	-	-	-
CNN without class weights	24	36.8	7.99	26	36.7	12.62	27	**31.3**	24.52
CNN with class weights	49	34.9	9.85	55	35.7	14.94	67	30.8	**26.86**
CNN-FE	58	36.9	9.4	65	36.7	14.11	70	31.1	24.76
CNN-FE + FastText	53	**37.3**	10.0	59	36.8	15.47	63	30.6	26.45
FFNN (128,64,32,71) with pre-processing	51	**35.57**	9.13	59	**35.31**	14.22	**71**	**29.64**	26.75
FFNN(256,128,64,71) with pre-processing	**68**	35.11	**11.76**	69	34.88	**17.76**	**71**	28.92	28.88
FFNN(128,64,71) with pre-processing	**68**	34.83	11.18	69	34.6	17.07	**71**	28.94	**31.19**
LSTM(128 lstm, 128 lstm, 32 dense, 71 dense) with pre-processing (50 epoch)	67	33.59	10.37	70	33.76	15.89	**71**	28.46	27.79
LSTM(128 lstm, 64 lstm, 71 dense) with pre-processing (150 epoch)	65	34.15	11.11	69	34.74	16.89	**71**	29.37	30.14
LSTM(128 lstm, 64 lstm, 71 dense) with pre-processing (200 epoch)	**68**	33.56	11.16	**71**	33.77	16.55	**71**	28.70	29.69

Table 5: Comparative Results with Related Works

From the table we can evaluate our comparative result analysis as follows:

1. Shaded region results are the results of related works.[1] The bold scores in this region represents the maximum value for the particular evaluation metric.

2. White region results are the results from our model. The bold scores in this region represents the maximum value for the particular evaluation metric.

3. We can see better results of TL and TR in our model from Table 05.

4. From Table 05 we can see that our model's F1 measure is a bit less but closer to the results of the related works.

5. After result extraction with different constraints (epoch number, layer number of FFNN/LSTM etc.) in our model we analysed that for a particular constraint when TL and TR increases then the F1 measure slightly decreases and on the other hand the constraints for which F1 measure value increases then the value of TL and TR slightly decreases.

5 Conclusion and Future Work

In our thesis work we proposed a model where we represent each plot synopsis as feature vectors which we create through pre-processing, tokenization and vectorization. Then with these vectors we input it in our feed forward neural network model and our LSTM model to get two sets of probability score of the tags. From that we get TL, TR and F1 measure values. Comparing with previous related works we have found slight improvement in TL and TR values keeping F1 measure value a bit lower than the results of the previous related works. Improvement in the result has been found due to pre-processing, tokenizing and vectorizing of data. But still there are future works that can be done from our thesis work such as:

- There is still a possibility that the accuracy of our work can be increased by applying different approach or model, or by twitching with some parameters of our work's model.
- In related works we see that one of the model used the flow of emotions with CNN to predict tags from movie's plot synopses.[1] But in our work we didn't consider the flow of emotion with our model. So integrating the flow of emotions with our model may give better result of evaluation metrics which can be a good future work.

References

[1] Sudipta Kar, Suraj Maharjan, Thamar Solorio. August 20-26, 2018. Folksonomication: Predicting Tags for Movies from Plot Synopses Using Emotion Flow Encoded Neural Network. Proceedings of the 27th International Conference on Computational Linguistics, pages 2879–2891 Santa Fe, New Mexico, USA.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Google AI Language.

[3] Rajaraman, Anand and Ullman, Jeffrey David. 2011. Mining of Massive Datasets. Cambridge University Press, New York, NY, USA.

[4] Jie Kang, Kyle Condiff, Shuo Chang, Joseph A. Konstan, Loren Terveen, and F.Maxwell Harper. 2017. Understanding How People Use Natural Language to Ask for Recommendations. In Proceedings of RecSys '17, Como, Italy, August 27-31, 2017, 9 pages.

[5] Bird, Steven, Edward Loper and Ewan Klein. 2009. Natural Language Processing with Python. O'Reilly Media Inc.

[6] DataFlair Team. 2019. Machine Learning Project – Data Science Movie Recommendation System Project in R. `https://data-flair.training/blogs/data-science-r-movie-recommendation/`

[7] Tony Yiu. 2019. Understanding Neural Networks. `https://towardsdatascience.com/understanding-neural-networks-19020b758230`

[8] Judit Ács's blog. `http://juditacs.github.io/2019/02/19/bert-tokenization-stats.html`

[9] Aditi Mittal. Understanding RNN and LSTM. `https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e`

[10] Kunal Gupta. Predicting Movie Genres Based on Plot Summaries. `https://medium.com/@kunalgupta4595/predicting-movie-genres-based-on-plot-summaries-bae646e70e04`

[11] Sudipta Kar, Suraj Maharjan, A. Pastor Lopez-Monroy and Thamar Solorio. 2018. MPST: A Corpus of Movie Plot Synopses with Tags.