Predicting movie genres using plot summary

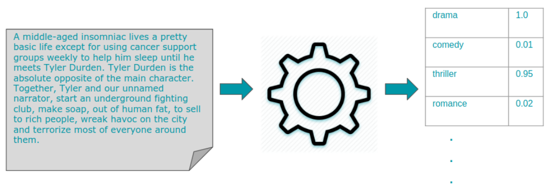
Neta Zinger, Open University
Dotan Asselman, Open University

Abstract

Text classification is a foundational task in many NLP applications. We decided to experience with text classification by building a model that given a movie plot summary, predict all of the movie genres. In this project, we examine the effect of training an RNN-based Language Model as a basis for learning representation of movie plot summaries. By learning those representations, we've been able to achieve state-of-the-art results in genres classification using one simple fully connected layer in a neural network. See our code at: https://github.com/dotannn/nlp-final.git

1 Introduction

In this project, we took upon ourselves the mission to predict and classify movie genres out from a movie plot textual summary. This problem is a sub-problem of text classification and since a movie can belong to several genres this considered to be a multi-label problem.



A middle-aged insomniac lives a pretty basic life except for using cancer support groups weekly to help him sleep until he meets Tyler Durden. Tyler Durden is the absolute opposite of the main character. Together, Tyler and our unnamed narrator, start an underground fighting club, make soap, out of human fat, to sell to rich people, wreak havoc on the city and terrorize most of everyone around them.

drama	1.0
comedy	0.01
thriller	0.95
romance	0.02

Figure 1- The task in high-level

Today, in leading website like IMDB and "rotten tomato", movie genre classification is done completely manual. This process is hard and expensive. Been able to automate this process, even if not 100% accurate, can save time and money. In another perspective, the fact that this process is currently done manually makes this problem highly suitable for applying machine learning algorithms, because of the huge amount of (already) manually annotated data. This way we can save time on annotating, and focus on the core algorithms.

As can be seen in figure 1, the input for the model is a movie plot summary, while the output is a vector with a probability for each genre. The final classification of the movie's genres is determined by taking the genres who are passing a certain threshold. Also, the genres probability vector before applying the threshold contains finer details about the movie (like how much of it is a Drama, Comedy, etc.). We believe this vector could be useful in finding similarities between movies. For example, for a movie recommendation system, similar to systems being used in companies such as Netflix. The probability vectors can be considered as points in a metric space, and similar movies will have low Euclidian distance from each other.

The problem of the classifying movie genres can be a good proxy problem for the problem we really wanted to solve, which is an automatic extraction of surgical intraoperative events from surgery post-operative textual notes. Basically, understanding which kind of events (errors & complications) surgery procedure may contain based on the notes. In this case, there is a shortage of annotated data, but we believe that the same algorithmic approach that is being used in our current problem can be used in the second.

2 Baseline method

We decided, as a baseline, to use the naive-Bayes algorithm. The naive-Bayes algorithm is being used as a baseline in several other papers (e.g. [3]) and is considered a classic method in text classification. The input of our model is a plot summary but the naive-Bayes require a feature vector as input, so we needed to do some preprocessing. We chose to use BoW (bag of words) as a feature vector.

The preprocessing composed of 3 steps:

- We first build a corpus that contains all the input plots that we have.
- For each word, we do a tokenization process (lower, remove -, 'and such).
- After we have our corpus, for each plot we build the feature vector by counting the number of each unique word in movie plot summary.

We still have a couple of problems in our BoW feature vector. First, we need to normalize the vector because some movie plots are longer than others. The normalizing is done by:

$$fv_{word} = \frac{count(word)}{lengh(summary)} \tag{1}$$

Second, we want to remove *stopwords* that do not contribute to the feature vector (and can even do damage). Words that are frequent in the English language e.g. the, is, an, etc. Now we can use naive-Bayes for the learning part. We chose to use the `sklearn` implementation of multinomial naive-Bayes.

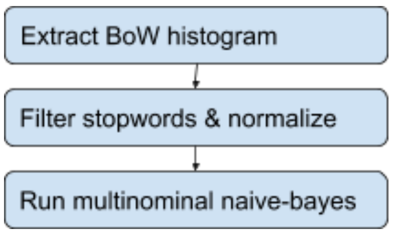


Figure 2- Baseline method steps

Pros:

- Fast and easy to implement and run
- Gives a good result on a small set of data
- Works well with a large dimension of a feature vector

Cons:

- When building the BoW we lose the or-der and context of words in the movie plot. This information can be helpful in our classification problem
- This naive-Bayes does not support multi-label out of the box
- In this implementation, we do not deal well with word that did not appear in the training set (this could be dealt with by using smoothing techniques).

3 Our Method

We propose a solution of using an RNN (recurrent neural network) which we will train in two stages.

In the first stage, we will train the network as a Language Model, there we will predict the next word in the sequence. In the second stage, we will change the *classification head* of the LM with a new head that will be in charge of predicting the movie genres.

The LM train stage serves as a preprocessing step, where we learn a better representation of a sequence of words. We are doing so by connecting the recurrent layers to a classifier which predict the next word in a text sequence. Training an LM this way allows us to use any textual data and have access to an almost unlimited amount of "annotated" data (because the labels are the next word in a sequence). After we are done with the LM training stage, we will save the layers that learned the textual representation (the *encoder*, see figure 3) and we will replace the classification head with new layers that will be in charge of predicting the movie genres.

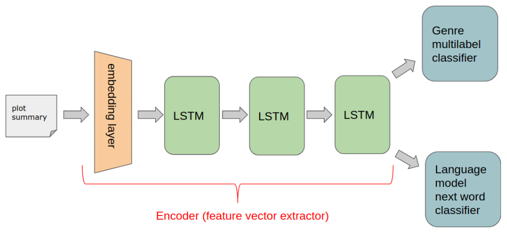

plot summary
embedding layer
LSTM
LSTM
LSTM
Genre multilabel classifier
Language model next word classifier
Encoder (feature vector extractor)

Figure 3 - Model architecture scheme

As can be seen in figure 3, the two network share the same architecture in the beginning, which we call the *encoder*. They both use an embedding layer followed by 3 LSTM layers. For LM classification head, we used an FC layer with `softmax` activation, so we can predict the next word. For the genres classification, we also used an FC layer, but with sigmoid activation, to support multi-label genres for each movie.

3.1 Design decision: Language Model vs word2vec

The idea behind word2vec is to learn a representation of a word in a way that correlates with the semantics similarities in a metric form. For example, in the representation space this equation will hold:

$$emb(\text{“}king\text{”}) - emb(\text{“}man\text{”}) + emb(\text{“}woman\text{”}) \cong emb(\text{“}queen\text{”}) \qquad (2)$$

The downside of this approach is the fact that word2vec representation doesn’t take advantage of the word’s context. Meaning, the words came before the current word in the sentence does not affect the word-vector. In contrast to word2vec, when we learn a representation using LM, the representation is not for each word independently, but to the whole sequence. The representation is a product of the *encoder* which contains 3 layers of LSTM hence, by default, influenced by all previous words in the sequence. The result of that is that even for the same word, it will have a different impact on the representation, depending on the word’s context. This way we achieve richer representation than word2vec, and a stronger basis for our classification process.

3.2 Design decision: on RNNs and LSTM

RNNs typed neural networks are highly useful in the field of NLP. The reason for that is because recurrent layers allow the network to learn a model not only based on current example, but also "remember" previous examples in the sequence. This way the returned output is depending on the history, or context, of current sample. In many cases of NLP tasks, we have a sequence of words and each word for itself is not enough, we need to take look at the sequence as a whole, and this is where RNNs shine.

More specifically, in our case, we used LSTM layers. LSTM are type of RNN layers with interesting internal process. Each LSTM layer accept 3 inputs:

- **Cell-state:** c_{t-1}, inner value that we get from the previous run t-1.
- **Hidden-state:** h_{t-1}, the output of the previous run t-1.
- $\boldsymbol{x_t}$: the current sample

The internal structure contains several inner gates:

- “forget gate layer”: decide how much of the cell state to keep. Meaning, how much of the history to save (how much to forget)

$$f_t = \sigma(W^f x_t + U^f h_{t-1}) \qquad (3)$$

- the second gate gets the current input x and the hidden state from t-1 and its purpose is to calculate a new delta for the cell state and the delta weight (calculate how important is this update to cell state)

$$\bar{c}_t = \tanh(W^c x_t + U^c h_{t-1}) \qquad (4)$$

$$i_t = \sigma(W^i x_t + U^i h_{t-1)}) \qquad (5)$$

- The third gate use the outputs of first two gates to calculate the new value for the cell state.

$$c_t = i_t \odot \tilde{c}_t + f_t \tilde{c}_{t-1} \qquad (6)$$

- The last gate is for determine the value of the output ht. we will do so by calculating how much of the new cell state we want to copy to the output after normalizing the cell state to the range of -1 to 1 using `tanh` function.

$$o_t = \sigma(W^o x_t + U^o h_{t-1)} \qquad (7)$$

$$h_t = o_t \odot \tanh(c_t) \qquad (8)$$

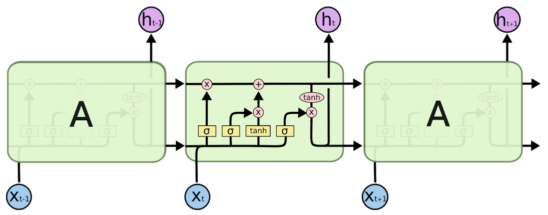


Figure 4 - Inner structure of LSTM layer

For each one of the 3 LSTM layers that we used, we worked with a modern implementation of LSTM called AWD-LSTM[2]. The different between AWD-LSTM and vanilla LSTM is the use of dropouts in the hidden-to-hidden connections of the LSTM. Meaning that, and based on the mathematical description of LSTM (equations

3-8), the dropouts are being applied on matrices: U^i, U^f, U^o

In addition to AWD-LSTM dropouts, we also use standard dropouts between all the layers in our network, for regularization purposes.

3.3 Design Decision: Dropouts for regularization

The main idea behind dropout is to randomly drop some of the outputs from a layer of neural network. This results to a network where in each layer more nodes learn the same features as a redundancy (to compensate on others that were dropped). This redundancy allows us to handle with partial information and improve the robustness of the model.

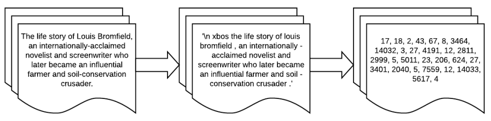
4 The training process

The training process is composed by several states.

4.1 Preprocess: Indexing the input

We did several preprocessing steps:

- We added a start word (“`xbos`”) to every movie plot summary.
- We removed any metadata from the plot summary such as html tags (can be seen in function `fixup`()).
- We lowered all the input data letters.
- We performed tokenization with `scapy` library.
- We indexed all the words in the input by taking the 60,000 most common words in our corpus, and mapping the rest to “`UNK`” (with index 0)


The life story of Louis Bromfield, an internationally-acclaimed novelist and screenwriter who later became an influential farmer and soil-conservation crusader.

'\n xbos the life story of louis bromfield , an internationally - acclaimed novelist and screenwriter who later became an influential farmer and soil - conservation crusader .'

17, 18, 2, 43, 67, 8, 3464, 14032, 3, 27, 4191, 12, 2811, 2999, 5, 5011, 23, 206, 624, 27, 3401, 2040, 5, 7559, 12, 14033, 5617, 4

Figure 5 - Indexing preprocessing steps

By the end of the preprocess each plot summary was represented by a sequence of indexes.

4.2 First step: Language Model training

In order to train the LM, we used as an input randomly length sequence of words. The mean sequence length was the value of the parameter BPTT (back-prop through time, we set to 70), we did so to avoid running the same sequences repeatedly in each epoch, as done in [2].

In addition, we check the possibility of training the network without dropouts. As can be seen in figure 6 around epoch 16 the loss of the validation starts to go up (while the loss of the train keeps going down) this insight suggests that our network start to overfit.

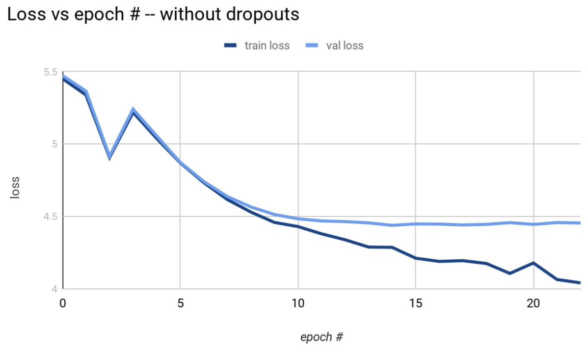


Figure 6 - Loss over time w/o dropouts

As can be seen in figure 7, when we use dropouts the validation loss is even slightly better than the training loss. This happens because the dropout is being disabled in the validation steps.

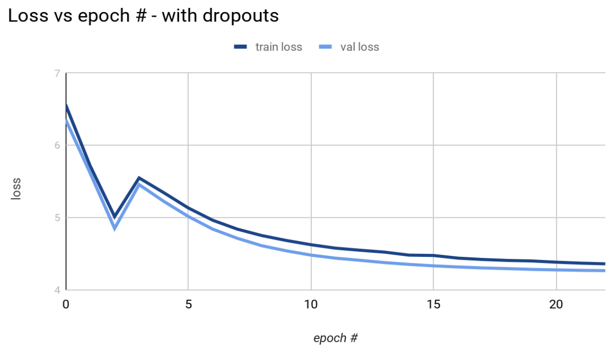


Figure 7 - Loss over time with dropouts

4.3 Second step: Classifier training

The classifier trained in 3 steps. Because we are basing the classifier on the result of the Language Model in each layer of the encoder, we start with a fine-tuning iteration, where we freeze the weights of all the encoder layers and train only the classification layer. The reason for doing this step is to avoid damaging encoder's representation quality learned by the LM. We are doing so for one epoch, then we "unfreeze" the last encoder layer and train again for one epoch. The last step is to "unfreeze" the whole network and train again for several epochs (about 25 more epochs).

This approach gives us more robustness to the training process and maximizes the pre-training of the Language Model.

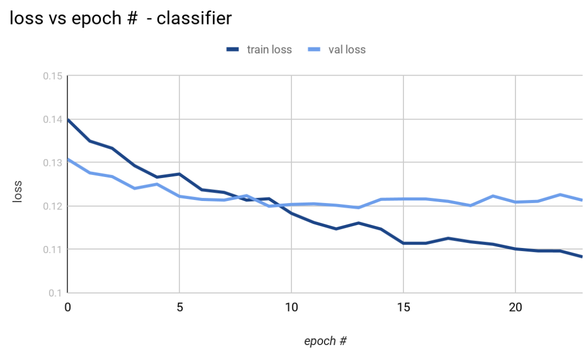


Figure 8 – Classifier loss over time

4.4 Design decision: Learning rate strategy

We used Cyclic Learning Rate (CLR) technique that was also used in [1]. The intuitive learning rate strategy approach is to lower the learning rate as we proceed in the training. Counterintuitively, when using CLR, we change the learning rate cyclically between predetermined high and lower bounds, as can be seen in figure 9.

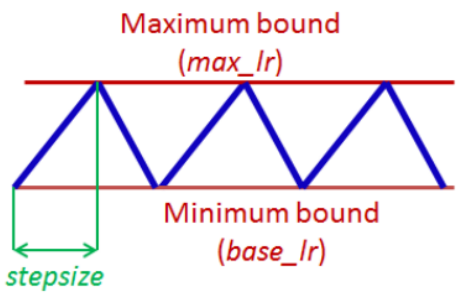


Figure 9 - Learning rate selection with CLR

The theoretical basis in this approach is the observation that the major challenge in converging to global minimum is the fact that the optimizer often stuck in local minima or even more commonly, a saddle point. By raising the learning rate from time to time the optimizer gets a “friendly push” to get through those critical points while lowering the learning rate will make the optimizer stay there indefinitely.

5 The Dataset

We are using IMDb's open to the public movies dataset. We are crossing the plots dataset with genres dataset for each movie to get (plot summary, genres) couples. This brings us a total of 344,662 movie plots with 34 possible tags(genres) while each movie can be tagged with more than one genre. The distribution of genres is of course not uniform. The most common genres are “Short movie” (40%), “Drama” (38.79%) and "Comedy" (23.32%). There are also genres that were very rare (have only very few samples in our dataset) such as "Lifestyle", "Sex" and "Hardcore". We decided to filter out genres with less than 10 samples in our dataset. Also, we decided not to handle short movies and focus on full-length movies hence filtering out 40% of our dataset. After the filtering took place we had 206,766 movie plots with 27 possible genres.

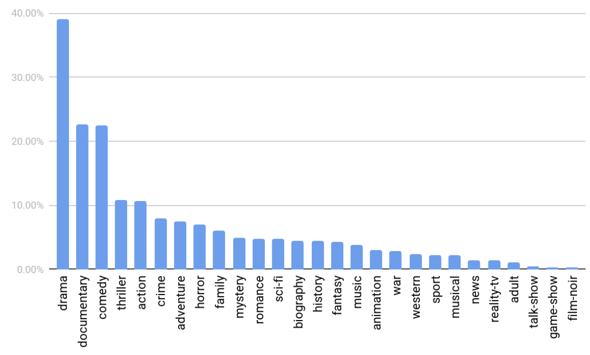


Figure 10 - Genres distribution over the dataset

Figure 11 shows the distribution of the number of genres per movie. Notice that about third of the movies have only one genre (32.7) but some have more than 4 genres.

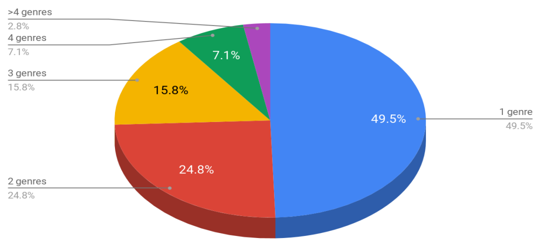


Figure 11 - # of genres for each movie distribution

We randomly divide the dataset for *training set* (90%) and *validation set* (10%).

6 Evaluation metrics

The main evaluation metric that we used is the `Jaccard Index` (IoU). This metric is calculated by the ratio of the number of genres that were truly predicted divided by the total number of predicted + true genres. We use `P` to denote predicted genres, and `T` to denote ground-truth genres.

$$Jaccard = \frac{|P \cap T|}{|P \cup T|} \quad (9)$$

In addition to the Jaccard index, we calculated 3 more standard metrics:

$$Precision = \begin{cases} \frac{|P \cap T|}{|P|}, if\ |P| > 0 \\ 0, otherwise \end{cases} \quad (10)$$

$$Recall = \frac{|P \cap T|}{|T|} \quad (11)$$

$$f1_{score} = \frac{2\, precision * recall}{precision + recall} \quad (12)$$

Where each metric has its own limitations. For example, Recall metric will get a perfect score for a naive model that will return ALL genres for any plot summary it gets. In the opposite direction, the Precision metric will get a perfect score if was correct only in one genre each time.

7 Experiments results

We split the results section to 3 sub-section. In the first, we examine the effect of dropouts on LM and classification results. On the second, we show several experiments we made in order to select the best configuration for the model. The third sub-section is a comparison of our best results with the baseline method.

7.1 Dropouts influence on models

We demonstrated the influence of the dropouts on the Language Model (predicting the next word) and on the classifier

Method	Accuracy(LM)	Jaccard(Classifier)
With dropouts	0.2988	**0.556**
Without dropouts	0.2777	**0.452**

Table 1: Effect of dropouts.

7.2 Optimizing configurations

We have tested several configurations for embedding size and hidden activation in the LSTM layer to find the one returning best results.

Configuration (emb, h_act)	Jaccard	Precision	Recall	F-score
(250, 640)	0.556	**0.726**	0.63	0.634
(350, 640)	0.551	0.718	0.633	0.632
(250, 720)	**0.557**	0.723	**0.642**	**0.638**
(350, 720)	0.557	0.724	0.639	0.637
(400, 1150)	0.554	0.724	0.632	0.633

Table 2: Classifier results on different configurations.

As we can see in Table 2 we received the best result on embedding size of 250 and hidden activation size of 720 for each LSTM.

7.3 Compare to baseline

After optimizing our result, the final step is to compare our results with the baseline:

Method	Jaccard	Precision	Recall	F-score
Baseline	0.355	0.543	0.355	0.405
Ours(Best)	**0.557**	**0.723**	**0.642**	**0.638**

Table 3: Best results compared to baseline.

As Table 3 shows, our proposed solution to the problem produce far better result than the baseline, and even improve the best result we have seen so far [3]

8 Conclusion

In this project, we present a different approach for classifying movie genres from movie plot summary, by using RNN based Language Model to learn a representation of the plot summary, then run a classifier network to learn to predict the genres.

As can see in [3], and also from our final results, this problem is far from being a "solved problem". Although we did manage to significantly improve the results for Jaccard index, reaching above 55%, there is still a lot of work to be done in with this task.

In this project, we learned the importance of representation and how LM is useful for learning representations for word sequences.

We also learned about the effect of dropouts, and how important it is to use dropouts, for a model to generalize well.

We believe that more improvements can be done. One promising approach would be to add a pre-training stage for the Language Model on a bigger dataset such as WikiText-103, as can be seen in [4]

In the future, we would like to test the transferability of this method to other domains. Specifically, we want to try extracting key events (errors & complications) from surgical post-operative reports.

References

[1] Leslie N. Smith. 2017. Cyclical Learning Rates for Training Neural Networks. U.S. Naval Research Laboratory.

[2] Stephen Merity, Nitish Shirish Keskar, Richard Socher. 2017. Regularizing and Optimizing LSTM Language Models. Salesforce Research, Palo Alto, USA.

[3] Quan Hoang. 2018. Predicting Movie Genres Based on Plot Summaries. University of Massachusetts-Amhers

[4] Jeremy Howard, Sebastian Ruder. 2018, Universal Language Model Fine-tuning for Text Classification. University of San Francisco, Insight Centre, NUI Galway Aylien, Dublin.

[5] Data source: ftp://ftp.fu-berlin.de/pub/misc/movies/database/frozendata/