EXPLORING METHODS FOR PARSING MOVIE SCRIPTS - FOR SOCIAL INJUSTICE ANALYSIS

Anonymous authors
Paper under double-blind review

ABSTRACT

When it comes to analysing movie scripts for things like bias and given the variation of movie script formatting due to inconsistencies by the authors, it is important that we create methods that can help extract all the relevant features required for any further analysis. In this paper, we discuss multiple parsing techniques that can be used to extract features and understand the structure of movie scripts in an automated fashion. We compare and contrast the accuracy and time of a rule based and a variety of machine learning approaches including; Deep Neural Networks, Decision Trees and BERT for sequence classification model, for film script parsing.

1 INTRODUCTION

Tackling social injustice is a major challenge for media organisations around the world Cho & Johnson (2020), Chiazor et al. (2021). Enabling automatic processing of media content, such as news articles, game dialog, movie scripts and books, would enable a variety of artificial intelligence techniques to be applied to this challenge. In this paper, we focus on the task of natural language parsing of media content, with the aim of enabling a variety of downstream tasks, such as applying social injustice tests. For example, the Bechdel test Bechdel (1986) is able to identify sexism within movie scripts. The idea is to enable tests such as these to be included in journalist and media workflows, with the aim to reduce the likelihood of social injustice in the media, or at least, alert it to a journalist. If these, and other downstream tests are to be executed at scale, and included in journalist workflows, it is crucial to be able to perform these tests automatically, by parsing large quantities of natural language scripts correctly. Therefore, in this paper, we take a first step towards automatic parsing of media content, by evaluating a variety of techniques for natural language parsing of movie scripts, given the availability of movie script data.

Our first aim was to be able to perform automated testing on a large corpus of movie scripts, and we chose the IMSDB The Internet Movie Script Database.While analysis of this database has been done before, Lee et al. (2017) did not attempt to classify lines and while Winer & Young (2021) did attempt to do this, they encountered issues. They were only able to process around 92% of scripts, and even within the 92% they encountered errors. This is understandable as can be seen from the raw HTML in Figure 1, the only information we have to tell one line from another is <b> tags, and the indent (leading spaces or tabs). However, there is no consistent usage of this syntax across different movie scripts. A non-exhaustive list of such issues is provided in Appendix Table 4.

We initially observed some general rule patterns governing a script's structure (e.g. a dialog is indented less than a characters name etc.), but for any rule we observed and created - there was a significant portion of scripts that broke that rule. In addition to this, scripts are not always internally consistent, i.e. indenting or styling rules may change within a script. For example, scripts were missing <b> tags at random, or they changed indent for the same line type, or used the same indent for multiple line types, hence the trouble encountered by Winer & Young (2021) in creating a set of rules and grammar to parse these scripts. The lack of consistency meant simple rule-based parsers were not accurate.

Being able to parse any script to a high level of accuracy is critical. For our first example of social justice test, we intend to apply the Bechdel-Wallace test Bechdel (1986). In order to do so, we need to extract the characters and dialog from the scripts, assign them genders and assess who is speaking

```
<b> PROFESSOR </b>
" Here.. He points to the Five Elements. "
<b> PROFESSOR </b>
" A weapon against evil. Amazing! I am going to be famous. "
<b> PRIEST </b>
" Then let us toast to your fame! Here Billy.. The Priest hands Billy a cup. "
<b> PRIEST </b>
" Drink! "
<b> PROFESSOR </b>
" To fame.. salud.. The Professor raises the cup to drink, and then... "
<b> PROFESSOR </b>
```

ß

Figure 1: Example HTML from movie script "The Fifth Element"

to whom. This separates us from some groups such as Lee et al. (2017) who were able to simply ignore scripts that broke their parsing systems.

In our case, it may be that a script is an excellent example of passing or failing the Bechdel test. We may also wish to analyse the collected works of a screenwriter or production company, therefore we are required to identify a method to parse any given script. The Bechdel test requires; First, two named female characters. Second, that they speak to each other. Third, that when speaking the subject is not a male character. As the test requires a single *instance* of proof to pass, it can pass or fail on single lines within a movie script. An example of a script which passes based on a technicality, which is not uncommon, is "Weird Science" Weird Science - Bechdel Test Movie List. As such, if the few lines that provide the pass were parsed incorrectly by our system, we would return an incorrect result. Therefore, in this paper we have attempted to explore multiple techniques for parsing any given script, to the best level of accuracy possible.

For the purposes of application, we will consider the elements laid out in Table 1 to be possible elements of a movie script.

Type	Description	Example
SCENE _BOUNDARY	The start of a Scene	INT. SURGICAL CHAMBER - NIGHT
SCENE _DESCRIPTION	Description of the coming scene	A group of black-clad ALF Activists, all wearing balaclavas,move down a corridor.
CHARACTER	Name of Character that is about to speak	CHIEF ACTIVIST
DIALOG	Speech	I can pop these, no problem.
DIALOG _DESCRIPTION	The Description of the coming dialog	(BLURTS)
WHITESPACE	Empty lines	
META	Other data, page numbers etc..	Deadpool Final Shooting Script 11/16/15

Table 1: Feature Types Expected

2 Background Literature

Analyzing Movie Scripts as Unstructured Text Lee et al. (2017) looked at parsing the scripts from the IMSDB with the goals of "understanding patterns and narrative flow that can be present in storytelling." The parsing they did in their work was only suitable for the sentiment. We will be extracting a more complex structure from the script.

Parsing Screenplays for Extracting Social Networks from Movies Agarwal et al. (2014) also looked at the IMSDB- and again encountered problems using rule-based systems to parse scripts,

noting similarly that many scripts are not well structured. They also explored an ML approach, using SVM as the model type, and a similar list of features to us. In the end, they gained a 69% accuracy on average with their 8 models. However, they were able to reach 96% accuracy using a Maj-Max ensemble model. We hope to experiment with other ML model types and see if we exceed this.

Automated Screenplay Annotation for Extracting Storytelling Knowledge In Winer & Young (2021), the authors make some excellent progress in parsing IMSDB scripts. Whilst they did not break the script into as many elements as we do, they had issues parsing large numbers of scripts (around 8% of what they collected were unusable due to formatting issues described earlier), whereas we would hope to be able to parse any given script. The use of RDP (recursive descent parser) is interesting, however they require particular markers to be there such as the shot type and Time of Day. We hope to pass any script even if normal markers are missing.

3 METHODOLOGY

3.1 RULE BASED

Naively, before realising the scope of the problem, we attempted a simple rule based line parser. An example rule, *If* `<b>` *bracket in line, and indent is less than X and greater than Y tag the line as a 'DIALOG'*. This worked almost perfectly on the first script we attempted, it failed on further scripts and we moved the X/Y boundaries to try and accommodate more scripts. Eventually, it was realised that certain boundaries had no overlap between scripts and no universal set of rules could be formed. Our first solution was to allow a custom set of bounds to be passed by a user for each script. This fixed the problem but was not suitable for automatically parsing scripts from a large corpus as we intended. We can anecdotally report that for some scripts, this was very accurate and made few mistakes, but for more than half of the scripts we initially tested on had issues with classifying the lines.

3.2 MACHINE LEARNING – USING IBM WATSON

As a first machine learning approach, we used the IBM Watson Natural Language Classifier service [1] [2]. It is an enterprise solution to train, customise and deploy natural language classifiers at scale. We uploaded our own set of training data to the service and had it build a black box model using the labels we provided. The accuracy we obtained from using this approach in pre-testing was below 80%. In addition, REST API calls were needed for each line. As such, we did not continue to use this parser and it does not appear in our results section.

3.3 MACHINE LEARNING –DEEP NEURAL NETWORK

Our assumption for the performance we observed from the Watson system is that, it did not have enough access to the features required to tell lines apart. Our next approach was to create an NLP model of our own. When creating the training data for the Watson system, we had to classify the lines ourselves. Using this experience, and looping over confusion matrices from past models, we developed a set of line and word based features, listed in full in **DNN Features**

Creating these set of features, improved our accuracy levels to around 90% (in fitting). The largest confusion remained in dialog vs. scene descriptions. As humans when a line was isolated from context, we would also struggle in this case, it made sense a DNN would have the same issue. For example, from "28 Days later,these are `DIALOG` which could have been a `SCENE_DESCRIPTION`: *"He's not a doctor. He's a patient."*, *"Top cupboard."*, *"And there's two hundred flats here. Most of them have a few cans of food, or cereal, or something."*

Please see Appendix for technical details on how our DNN was build - **DNN Technical details**

[1] https://www.ibm.com/cloud/watson-natural-language-understanding

[2] https://www.ibm.com/docs/en/opw/8.2.0?topic=ui-natural-language-processing-services

3.4 MACHINE LEARNING – DOCUMENT AWARENESS

As discussed in the last section we found it hard to correctly label lines out of context. When looking at the lines the model was most likely to classify incorrectly, we found they were some of the hardest for us as humans to classify as well. To fix this we added two features.

- `last_line_type` A one hot encoded representation of what the model thought the type of the last line was. In many ways, mocking a RNN or LSTM architecture.
- `relative_indent` Going back to the rule-based parser, while it was true that many scripts broke the formatting, the indenting levels relative to each other were often consistent.

Adding these features raised the accuracy (during fitting) to 96.3%. As you will see later on, this did not translate well into real world accuracy as seen in Table 2. In testing, this dropped to 82.4%, worse than the 87.0% offered by the the non-document aware based parser. We assume the reason for this, is because the parser was trained on training data which included a `last_line_type` feature manually added by human from the line above, which would nearly always be accurate. However, in real world application the `last_line_type` was actually populated based on the *predicted* class of the line before it , which might naturally be inaccurate given our model does not have 100% accuracy. So we suspect in training the parser weighed `last_line_type` feature highly, as it should have been a good predictor for classification. However, when real world data contained mistakes, this would cause misclassifcation. Worse, this would then chain into the next line and so on. We hope to address this by re-training the model on noisy data, and also by including confidence as a feature.

3.5 DECISION TREE BASED APPROACH

We used our existing training data form the **Machine Learning – Document Awareness** training to build Decision Tree based parsers. These were built using the `sklearn.tree DecisionTreeClassifier`. Data was encoded and scaled in the same manor as in the DNN models.

We then used hyper-parameter optimization to generate trees with: **Depth Range:** [1-10] **Criterion:** 'gini' or 'entropy' **Max Features Range:** 1-500

This resulted in two trees; The first a line based parser without document based features which reported at fitting time an accuracy of 98.6%. The second a document based parser with document based features which reported at fitting time an accuracy of 97.3%. The parameters for both are in the **Appendix - Decision tree Hyper Parameters**

3.6 BERT MODEL APPROACH

Language representation models like BERT (Bidirectional Encoder Representation from Transformers) Kenton & Toutanova (2019) have proven to be simple yet powerful for a wide range of natural language processing tasks. By fine tuning a pre-trained BERT model with one additional output layer, state of the art models can be created for sequence classification tasks. For this approach, we fine tune a BERT model for our given task, leveraging a `BertForSequenceClassification` model from the huggingface transformers library[3]. This model is a Bert Model transformer with an additional linear layer on top of the pooled output. Our goal was to obtain a model that can classify each line content in the script belonging to one of our potential classes. We train this model using the following steps:

We established a dictionary that represents each potential class in our training data. represented as

```
{'character': 0,'dialog': 1,'scene_description': 2,
'scene_boundary': 3,'dialog_description': 4,'meta_data': 5}
```

Replaced all labels in the training data using the values of the given dictionary. Using a train to validation split of 0.85 : 0.15, we obtained the training and validation sets. We encoded each

[3]https://huggingface.co/docs/transformers/

line content in our split sets using a `bert-base-cased` tokenizer config. The decision to use the 'cased' version as opposed to the 'uncased' version of the tokeniser is due to the observation that in varying movie scripts certain classes have the feature of been capitalised e.g. in the movie script (28 days later)[4], we observed that both characters and scene description tends to be capitalised. We pad each encoding to a max length of 218, and set the option of adding special tokens to true. The `batch_encode_plus` method was used (which is similar to using the tokenizer's call method directly). We used a pytorch TensorDataset to obtain a dictionary of the `input_ids`, `attention_masks` and the `labels_tensor`. Then we used a Dataloader which combined the training tensordataset with a *RandomSampler* and the validation tensordataset with a *Sequential-Sampler*. For our optimiser, we use the *AdamW* optimiser from the transformers library and set a learning rate (lr) of *3e-5*, with an epsilon (eps) of *1e-8*. We also use a linear scheduler with no warm up steps and we set the number of training steps to be the length of the training set multiplied by the number of epochs. During training we apply gradient norm clipping using pytorch's `clip_grad_norm` method.A cross entropy loss was computed per batch for each epoch, and the average training and validation losses obtained. Evaluation metrics: in addition to computing the average loss, we also used a weighted average `f1_score` and accuracy per class score to evaluate how the fine-tuned model performed. We obtained the probabilities of a line content belonging to each class by passing the model logits through a softmax function. Then we chose the class label with the highest probability.

Training the model with 4 epochs on a machine using 4 CPUs and 1 GPU core was sufficient enough to observe good results. With a sample size of over 4000 line content, the total time used was about 25 mins. Obtaining a weighted f1 score of about 0.996, average training loss of about 0.036 and average validation loss of 0.021 in the 4th epoch.

3.7 Hybrid Approach

The time taken to run the ML models with document awareness 3.4 or DNN 3.3, when not using high computation resources like GPUs or CPUs was around 20mins per script, which is significantly longer compared to the time it takes the rule based parser 3.1 to run. A compromise can be reached, where a user can examine a script and if it has both; consistent indenting, and no overlapping indent (lines of differing type don't share an indent). We can run one of our ML models on a small sample of the data, and use these classified lines (correct on average if we take a significant sample) to work out the indent levels and pass them to our rule based parser. The hypothesis here is that this will give the accuracy of the ML approaches while only taking about 2-3 minutes per script.

4 Results and Discussions

4.1 How we tested the approaches

We used the following list of movies from the IMSDB: *Deadpool*[1]*, 28 days Later*[2]*, Dune*[3]*, Alien*[4]*, The Martian*[5]*, Fifth Element*[6]*,Joker*[7]*,Blade*[8]*, Spider-Man*[9]*, Matrix*[10]

We took the first 500 lines of these scripts, we parsed this, and then hand corrected each of them so that we had 500 lines from 10 scripts with known good labels. For custom bounds, we used an approach of measuring the indent of the first instance of each line type, as we thought a user might. We then ran each of the parsers once (Only the hybrid approach has any degree of stochastic variance) , and collected the following information:

- Overall Accuracy
- Accuracy by line type
- Time taken to classify lines in milliseconds
- Overall time taken (including any pre-processing)

For an understanding of which parser names match to each of our methodologies please see Appendix - Parser Map

[4]https://imsdb.com/scripts/28-Days-Later.html

Parser	avg	stv
_line_ml	87.0	0.07
_line_ml_doc_aware	82.4	0.15
_line_rule_based	83.0	0.19
_line_rule_based-custom	51.2	0.35
_line_rule_based-smart	67.6	0.31
_line_dt_line	75.0	0.12
_line_dt_doc	68.2	0.1
_line_bert	**95.7**	**0.02**

Table 2: Parser Accuracy Data (avg=average overall accuracy, stv=standard deviation)

Parser	avg	stv
_line_ml	38816	1117
_line_ml_doc_aware	45062	1125
_line_rule_based	19	0
_line_rule_based-custom	18	1
_line_rule_based-smart	16033	2520
_line_dt_line	20816	500
_line_dt_doc	20965	495
_line_bert	**128175**	**2329**

Table 3: Parser Time Data (ms)

4.2 ACCURACY BY PARSER

Overall we seem to show excellent results, our only baseline being Agarwal et al. (2014) who achieved 69% accuracy max on a single model, which every one our parsers except line_rule_based_custom, and line_rule_based_smart show better results. We can also claim to match their ensemble model result of 96% with the single line_bert parser model. As observed in Table 2 the fine-tuned BERT approach line_bert scored the overall highest accuracy by a significant margin, and also showed very low variance in its performance. We would have hoped line_ml_doc_aware would have been persistently above the line_ml, and while it does often reach accuracy levels above 80%, line_bert is still the most consistent with high level of accuracy at about 95.7%. It is worth taking note of the fluctuations within the rule based systems. line_rule_based, line_rule_based_custom, and line_rule_based_smart each had a significant drop in accuracy on one or more movies.

Note: There is no smart indent data for some scripts, due to the imbalance of line types it could not create a set of bounds. In this case we record an accuracy and time of zero.

We noticed similar trends in Table 2, line_bert has the best overall accuracy, and lowest standard deviation, providing the most reliable and accurate results. We suspect the drop in accuracy we see with the line_ml_doc_aware compared to the 96% we saw when training is due to imperfect last_line_type in our test data compared to almost perfect data in the training data. Please see 3.4 for more thoughts on this. line_rule_based, line_rule_based-customs both show massive drops on The Fifth Element script, which we suspect is due to its indent levels being more inconsistent than most, showing the downfall of the rule based parsers, they are somewhat all-or-nothing.

More detailed data in Appendix - Full Parser Accuracy Data

4.3 TIME

In Table 3 and Figure 3 we can see the parsing times in ms for each of the scripts. It is clear that line_bert takes a significantly longer time than any of the other parsers, at an average of 128 seconds for 500 lines. We can say that in our experience, a movie script is in the range of 2500-3500 lines meaning a total parsing time of around 10-15 mins for script. Each of our other ML parsers

are within 20-40 second range for 500 lines. So we would expect a script to parse in 1:30min to 3:00 mins. We can also report from our attempts to optimise this that the majority of this time is spend in feature generation. `line_rule_based-smart` stands on its own in at an average of 16 seconds, due to its hybrid nature. We can see that the rule based parser takes the least time, coming in sub-seconds, showing were they may have an edge.

4.4 ACCURACY BY LINE TYPE

In Figure 4 we observe the overall accuracy by line type over all parsers.

We find the overall results in line with our expectations, though we note a few observations. Firstly `dialog_description` stands out as having significantly worse accuracy which was expected. We suspect this is an issue related to common formatting style we have seen in the scripts where `dialog_description` is split between two lines. A possible fix would be when a opening parenthesis is detected "(" to join any text to the current line that precedes the closing parenthesis ")".

Secondly is that `scene_description` has the highest overall accuracy (See Table 4 rather than `scene_boundary` which would have been our prediction due to its clear features (Low indent, capitalized, and contains "INT" or "EXT" in nearly all cases). Though looking at the data more closly we note in Table 14 an accuracy of 0% for `scene_boundary` using the `_line_dt_doc` parser, and suspect this has skewed our results.

With respect to notable differences on given parsers, `meta_data` is the only line type to drop below 90% on `_line_bert` which is to be expected `meta_data` is a catch all for any lines that don't fit within our model. We can see in the pure rule based parser in Figure 6 that `character` and `scene_boundary` are the highest at 88% and 99%, as shown in Table 8. Rule based parsers would seem to excel as we would expect, in situations where one or two clear features can determine the type. In Figures 6, 7 and 8 you can see that accuracy for one or more line types will often drop to zero. This outlines well the issues with these parsers, while highly accurate when the lines fall within the expected rules, they cannot cope with any deviation. This is particularly well shown in Figure 7 with custom bounds. Clearly the indent levels from the first few lines are not kept consistently though the scripts so our custom bounds often failed utterly. We can see in Table 8 the Smart indenter did much better than our custom indents in Table 7, though often not as well as the rule-based baseline. We will note that when the rule based system failed, i.e. The Fifth Element it did much better, adapting to a film that broke our rule set.

4.5 TIME VS ACCURACY

We observe that there is be a clear relationship between the time-cost and accuracy of any given parser, with what would appear to be diminishing returns. A line of `y=log(x)+c` would appear to fit. It is worth noting the significant outliers. `_line_bert` shows a significant increase in the accuracy,however the increase in time is even more significant, bringing its applicability into the range of batch possessing more than interactive tooling. `_line_rule_based` would appear to offer an excellent option, though as we have seen from the results above its variance in accuracy is the highest, and gives somewhat of an "All or Nothing" option. However given its speed, it should certainly be considered, especially in interactive applications in which an end user can detect if it has done poor job.

5 FUTURE WORK

Auto retraining

We hope our current system's user interface will allow human re-labeling of the parsed data. This will produce for us over time a large corpus of human annotated lines on which to re-train some of our ML model baselines. We suspect given more data, we could raise the general accuracy of the models, particularly through expanding the dictionary of common words by line type.

Characters as features

If a character's name in a line is preceded by an interjection, e.g "Hello Steve" is likely to be `DIALOG` and the past tense verb "Steve Ran" is likely to be `SCENE_DESCRIPTION` or present tense "Steve run!" is likely to be a `DIALOG`. We would however need to have a set of known characters to do this with. We would either do this via TMDB data or with a pre-run of a fast parser to extract the characters. Other new handcrafted features that we hope to consider in the future include; `Same indent as last line` would give us confidence that line types are the same. Scene descriptions are often multi-line. `Times we have seen an identical line` - This count would give a high probability towards a line being a character.

Feature Optimization Agarwal et al. (2014) removed sets of features and compared how they changed the overall accuracy of their models and on a per-category basis. We would look to do the same, as feature generation is one of the most expensive parts of our current process.

Probabilistic Indent Detector While our hybrid indent detector can perform better than our base rules or custom rules, it has drawbacks. We would suggest there is likely a similar distribution of line types within a script. i.e a script is on average X% Dialog, Y% Scene Description. If we count lines of the same indent we should be able to make a probabilistic guess as to which is which.

Sanity Checker We saw in Agarwal et al. (2014) the use of a sanity checker, we would like to reproduce and add this to our own system.

Other Parsing/ML Baselines

- **Recursive Descent Parser** - Winer & Young (2021) showed the power of this approach and we would like to see how well it works for us.
- **Naive Bayes** - Our common words feature showed a large jump in accuracy, so we would hope a Bayes-based approach would yield good results.
- **State Vector Machines** - Due to the promise shown by Grammar based parsers, and RDP's
- **LSTM/ RNN** - Given the sequential nature of our data (i.e this seems a clear choice to try as an additional baseline.

Ensemble Model

We saw in Agarwal et al. (2014) the power of Ensemble voting, as we can see in Table 6 and Table 9 certain parsers perform better on certain line types. Using a weighted voting system the overall accuracy of system could be improved. Especially if combined with the sanity checker giving a bias towards the linetype can come most logically next.

Creating and analysing tests based on the parsed data The ultimate goal would implement the ideas within Chiazor et al. (2021) i.e to to build automated tests that can use the parsed scripts, the annotations we build on top of them to answer questions on social injustices. For example, how are certain genders, sexuality's, races, creeds etc represented in film media content? can we detect and highlight bias at the level of a script been written, so as to encourage film studios to address such issues before filming even starts?

6 CONCLUSIONS

We have shown how the parsing of film scripts - as a means for extracting and classifying relevant features for further analysis is possible using both rule-based and machine learning based approaches. We have learnt from and discussed in particular what might make some ML based parsers not generalise as well as expected. However, we also discuss and show how a fine tuned BERT-based model is able to generalise and perform a lot better for the given task of parsing and classifying the line types in movie scripts, though at the cost of time increase. Currently, we offer all of the parsing options though a web based UI, as each option tends to offer some advantages depending on the given situation and trade offs a user is willing to accept. For now, users may choose a BERT based parser if they are willing to return later to the work, or a rule based one if they want quick results and are willing to correct possible mistakes. In the future we believe a combination of increased real world training data, sanity checking, and ensemble voting could produce a powerful parser that could handle scripts with a high degree of accuracy and at an acceptable time cost. We

would hope as well as script parsing and validation, to create and offer a data set for future analysis of social injustices.

7 Reproducibility Statement

Our scripts where scraped from thee IMSDB. Cleaned using a combination of Beautiful soup and RegEx. Our parsers where each take a single cleaned line as input and output a classification. Results where generated using first 500 lines of the above movies scripts. To recreate our classifiers the technical details are listed in the appendix, the authors would invite our peers to contact us, and we would be willing to share both our raw training data with labels, and datasets with generated features.

References

Apoorv Agarwal, Sriramkumar Balasubramanian, Jiehan Zheng, and Sarthak Dash. Parsing screenplays for extracting social networks from movies. In *CLfL@EACL*, 2014.

Alison Bechdel. *Dykes to watch out for*. Firebrand Books, Ithaca, NY, 1986. ISBN 9780932379177.

Lamogha Chiazor, Geeth De Mel, Graham White, Gwilym Newton, Joe Pavitt, and Richard Tomsett. An automated framework to identify and eliminate systemic racial bias in the media. In *AAAI Conference on Artificial Intelligence*, 2021.

Hyesun Cho and Peter Johnson. Racism and sexism in superhero movies: Critical race media literacy in the korean high school classroom. *International Journal of Multicultural Education*, 22(2):66–86, Aug. 2020. doi: 10.18251/ijme.v22i2.2427. URL `https://ijme-journal.org/ijme/index.php/ijme/article/view/2427`.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, pp. 4171–4186, 2019.

Seong-Ho Lee, Hye-Yeon Yu, and Yun-Gyung Cheong. Analyzing movie scripts as unstructured text. In *2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService)*, pp. 249–254, 2017. doi: 10.1109/BigDataService.2017.43.

The Internet Movie Script Database. The Internet Movie Script Database. `https://imsdb.com/`. (Accessed on 08/23/2022).

Weird Science - Bechdel Test Movie List. Weird Science - Bechdel Test Movie List. `https://bechdeltest.com/view/520/weird_science/`. (Accessed on 08/23/2022).

David Winer and R. Young. Automated screenplay annotation for extracting storytelling knowledge. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 13(2):273–280, Jun. 2021. URL `https://ojs.aaai.org/index.php/AIIDE/article/view/12994`.

A Appendix

A.1 Methodology

A.2 Parser Map

`_line_ml` - DNN with line based features only. See **Machine Learning –Deep Neural Network** above

`_line_ml_doc_aware` - DNN with Document based features See **Machine Learning – Document Awareness** above

`_line_rule_based` - Rule based Parser, with default settings (Our settings that generalise the best) See **Rule Based** above

`_line_rule_based-custom` Rule based Parser with custom settings. See **Rule Based** above

`_line_rule_based-smart` Ruel based parser, using DNN to create its settings. See **Hybrid Approach** above

`_line_dt_line` Decision Tree Parser with only line based features. See **Decision Tree Based Approach** above

`_line_dt_doc` Decision Tree Parser with additional Document based features See **Decision Tree Based Approach** above

`_line_bert` BERT based features. See **BERT Model Approach** above

A.3 Reproducibility

A.3.1 DNN Technical details

- Built using the Keras Library.
- Around 3500 Lines of Human labelled training data, with a 20/80 Test, train split.
  - Pre-classified using the rule-based method, and then human corrected.
- Dense Neural Network with dropout
  - Lines where capped at 100 words long, to give standard size
  - Around 1500 features (See Feature list above)
  - Network layers where 1500/750/750/750/325/200/6 with `relu` activation (and `softmax` for the output)
  - `sparse_categorical_crossentropy` was used to track accuracy and as the loss function.
- Data was transformed and we used One Hot encoding for most features.
- Standard Scalar was applied to non-encoded features. E.g. counts.
- Models stored in our custom file format. Which saved weights, scalars, features lists, enum mapping etc.

A.3.2 DNN Features

DNN Features - Lines:

- Line ends with Explanation Point
- Line Ends with Question Mark
- Line starts with parenthesis
- Total Count / percentage of line that is Arabic numerals in the line [0..9]
- Total Count / percentage of line that is punctuation
- Total Count / percentage of line that is ellipsis [. . .]

DNN Features - Words:

- Word is 1st /2nd/3rd Person
- Word is an interjection [gosh,yuck,anyhoo]
- Word is capitalised
- Word is Question [Who, What, Where,When,Why]
- Word is a base verb [love,drink]
- Word is a present/present participle verb [loving,drinking]
- Word is a past/past participle verb [loved,drank]
- Word is profane
- Word most commonly found in type:

  - This is warm encoded data from our own training data, it gives the relative frequency that the word is found in that line type. For example "hello" is mostly found in `DIALOG` and "gasped" is most often found in `DIALOG_DESCRIPTION`.
- Word is a contraction. [can't/don't/isn't]

A.3.3 Decision tree Hyper Parameters

Line Based:

```
        "hyper_params": {
        "criterion": "entropy",
        "depth": 9,
        "max_features": 451
},
```

Document Based:

```
        "hyper_params": {
        "criterion": "gini",
        "depth": 8,
        "max_features": 401
},
```

A.4 Tables

A.5 Figures

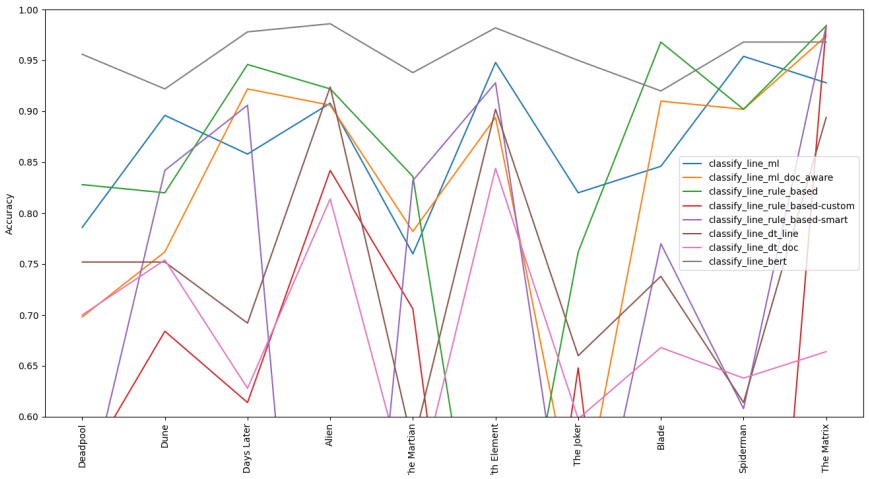

Figure 2: Parsing Time (ms)

Script	Lines	Issue
28 days Later	**SCIENTIST** (continuing; stumbling, **FLUSTERED)** These animals are highly contagious. They've been given an inhibitor.	Character Indent 26 spaces, Dialog Indent 11 spaces, Dialog Description at same indent as Dialog Dialog Description leaks into <b> tags
Dune	**EMPEROR** (quickly) A Third Stage Guild Navigator will be here within minutes! **REVEREND MOTHER** We felt his presence. **EMPEROR** I shall want telepathy during his visit and a report when we're finished.	Character Indent 25 spaces Dialog Indent 15 spaces Spacing not consistent with other scripts.
Fifth Element	**PRIEST (V.O.)** Amen..	Tabs not spaces.
Ghost busters	**WERECHICKEN (VO)** (Low, gutteral CLU-UCK) Maude looks over her shoulder AT CAMERA, reacts. **MAUDE** **(GASP!)**	Inconsistent use of <b> tags for the same line type
Alien vs Predator	**HIROKO** What kind of 'something'? CASSIE (O.S., onscreen) Easier if you come down and look.	Characters and Dialog at the same indent, and Inconsistent use of <b> tags for the same line type
Joker	**JOKER** I wasn't bothering her, I was-- **WOMAN ON BUS** (interrupts) Just stop.	Inconsistent use of indent for the same line type, 24 spaces vs 25

Table 4: Example Issues

Parser	Deadpool	Dune	28 Days Later	Alien	The Martian
ml	79	90	86	91	76
ml_doc_aware	70	76	92	91	78
rule_based	83	82	95	92	84
rule_based-custom	55	68	61	84	71
rule_based-smart	50	84	91	0	83
dt_line	75	75	69	92	57
dt_doc	70	75	63	81	51
bert	96	92	98	99	94

Table 5: Full Parser Accuracy Data

Parser	The Fifth Element	The Joker	Blade	Spiderman	The Matrix
ml	95	82	85	95	93
ml_doc_aware	89	49	91	90	97
rule_based	33	76	97	90	98
rule_based-custom	10	65	0	0	98
rule_based-smart	93	39	77	61	98
dt_line	90	66	74	61	89
dt_doc	84	60	67	64	66
bert	98	95	92	97	97

Table 6: Full Parser Accuracy Data

Type	avg	stv
character	0.81	0.06
dialog	0.71	0.18
dialog_description	0.32	0.12
meta_data	0.54	0.13
scene_boundary	0.74	0.08
scene_description	0.87	0.06

Table 7: Accuracy by Linetype across approaches

Type	avg	stv
character	0.88	0.31
dialog	0.81	0.34
dialog_description	0.0	0.0
meta_data	0.81	0.24
scene_boundary	0.99	0.02
scene_description	0.96	0.06

Table 8: Accuracy by Linetype(classify line rule based)

Type	avg	stv
character	0.37	0.41
dialog	0.6	0.44
dialog_description	0.0	0.0
meta_data	0.41	0.4
scene_boundary	0.59	0.51
scene_description	0.6	0.44

Table 9: Accuracy by Linetype(classify line rule based-custom)

Type	avg	stv
character	0.77	0.39
dialog	0.56	0.46
dialog_description	0.0	0.0
meta_data	0.51	0.43
scene_boundary	0.89	0.31
scene_description	0.77	0.37

Table 10: Accuracy by Linetype(classify line rule based-smart)

Type	avg	stv
character	1.0	0.0
dialog	0.86	0.06
dialog_description	0.65	0.36
meta_data	0.48	0.27
scene_boundary	0.97	0.08
scene_description	0.86	0.04

Table 11: Accuracy by Linetype(classify line ml)

Type	avg	stv
character	0.81	0.28
dialog	0.89	0.11
dialog_description	0.56	0.29
meta_data	0.4	0.26
scene_boundary	0.72	0.34
scene_description	0.94	0.06

Table 12: Accuracy by Linetype(classify line ml doc aware)

Type	avg	stv
character	0.87	0.13
dialog	0.51	0.47
dialog_description	0.03	0.05
meta_data	0.69	0.26
scene_boundary	0.76	0.38
scene_description	0.95	0.06

Table 13: Accuracy by Linetype(DT line)

Type	avg	stv
character	0.79	0.26
dialog	0.47	0.45
dialog_description	0.39	0.44
meta_data	0.31	0.26
scene_boundary	0.0	0.01
scene_description	0.98	0.02

Table 14: Accuracy by Linetype(DT document based)

Type	avg	stv
character	0.99	0.01
dialog	0.97	0.02
dialog_description	0.95	0.16
meta_data	0.74	0.25
scene_boundary	0.99	0.03
scene_description	0.93	0.06

Table 15: Accuracy by Linetype(classify line bert)

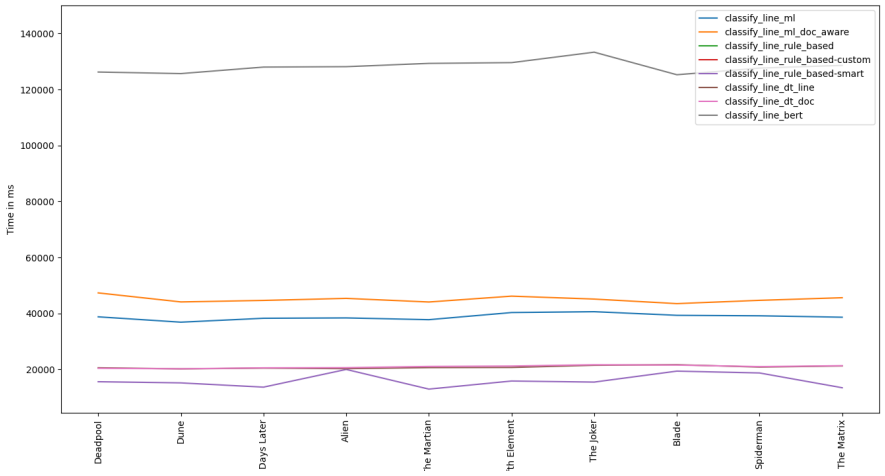

Figure 3: Parsing Time (ms)

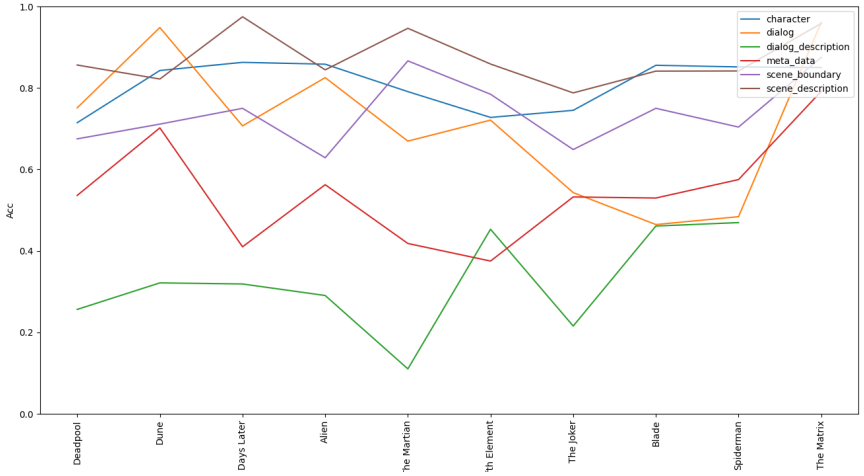

Figure 4: Accuracy by Line type

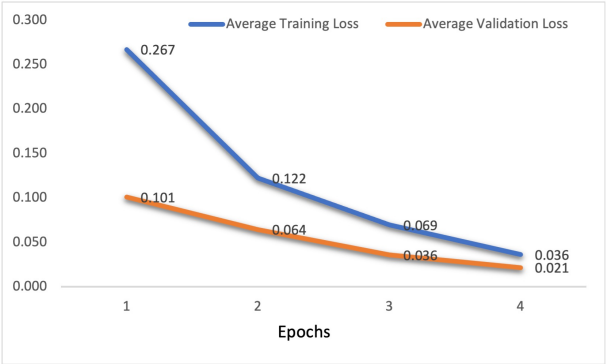

Figure 5: Bert Model Training Loss

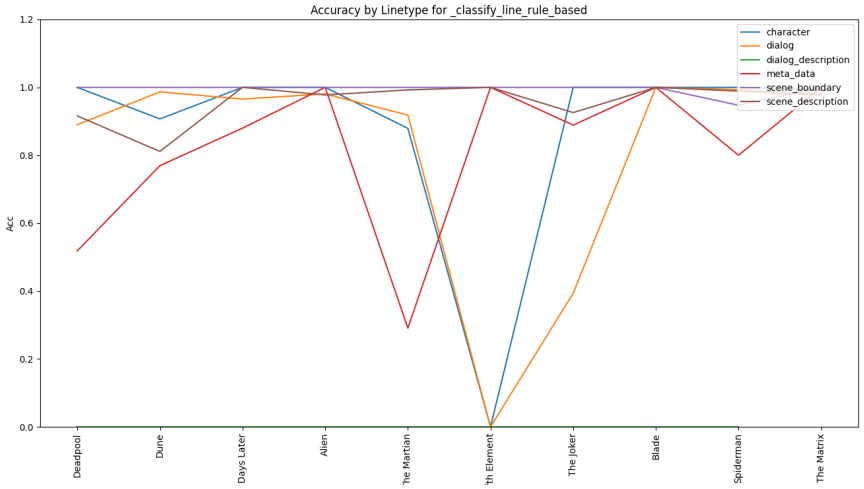

Figure 6: Accuracy by Linetype(Rule Based)

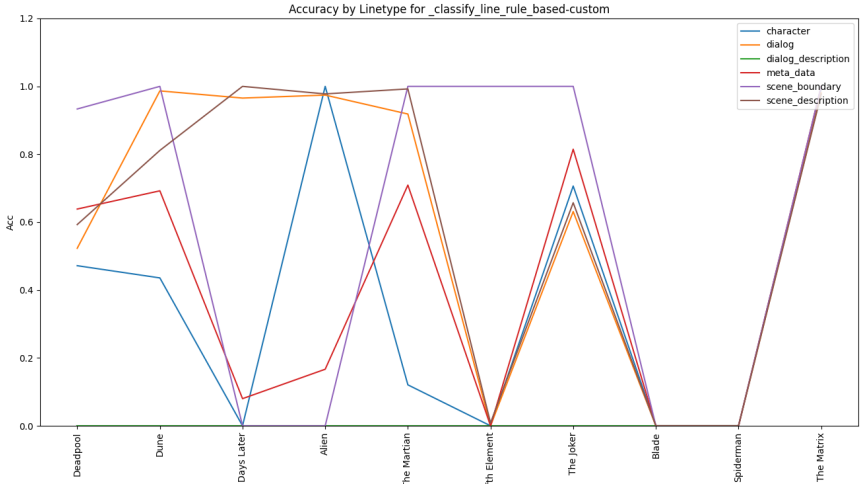

Figure 7: Accuracy by Linetype(Rule Based-Custom)

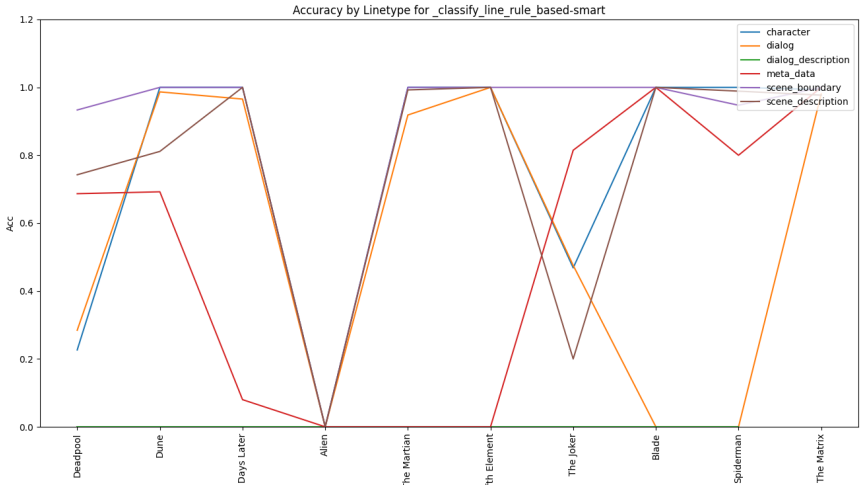

Figure 8: Accuracy by Linetype(Rule Based-Smart Indent)

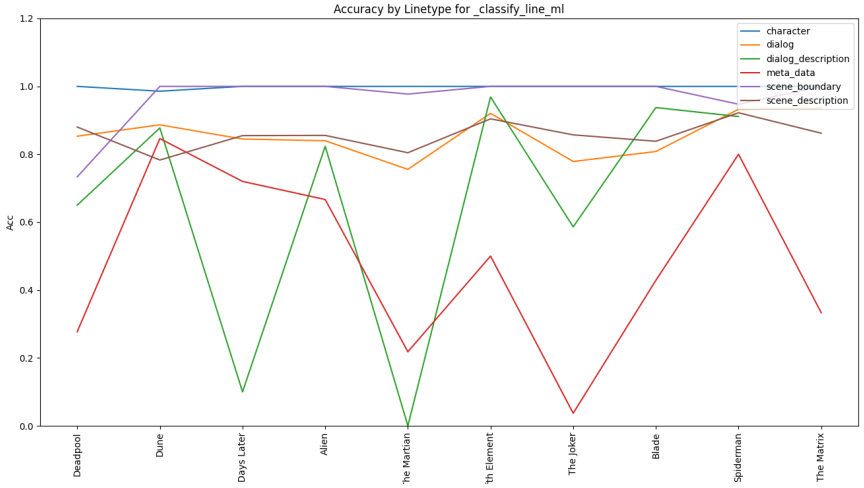

Figure 9: Accuracy by Linetype(Machine Learning)

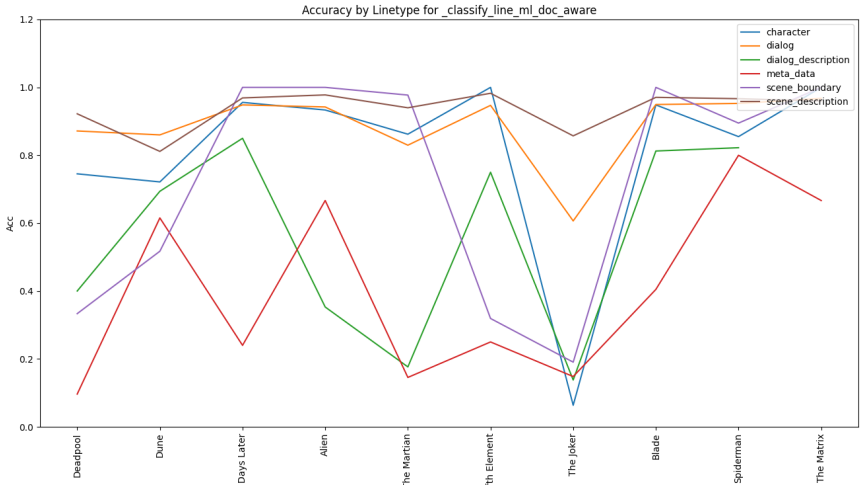

Figure 10: Accuracy by Linetype (Machine Learning Document Aware)

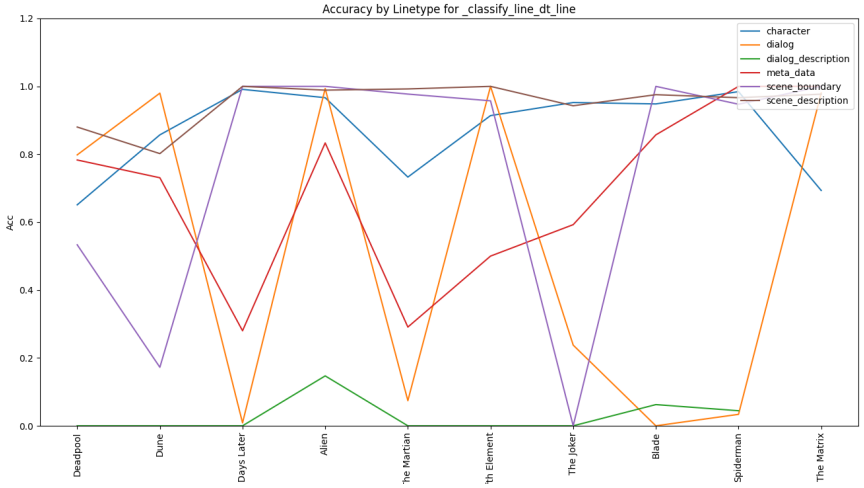

Figure 11: Accuracy by Linetype (DT Line)

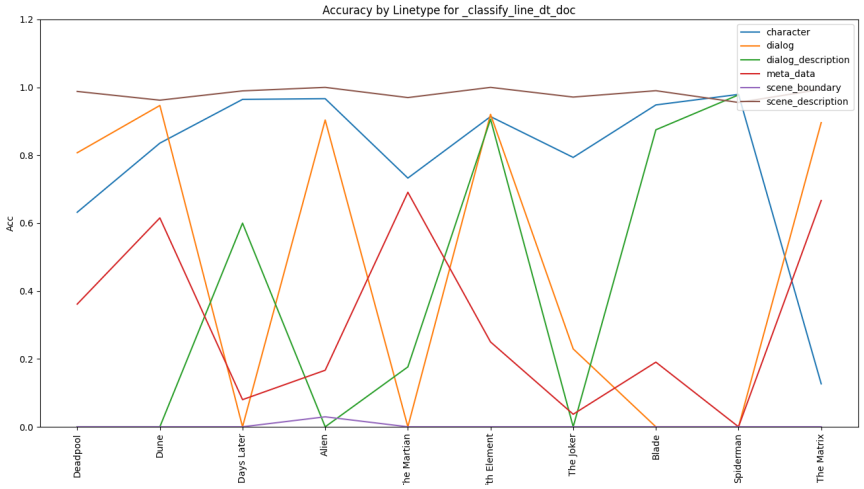

Figure 12: Accuracy by Linetype (DT document based)

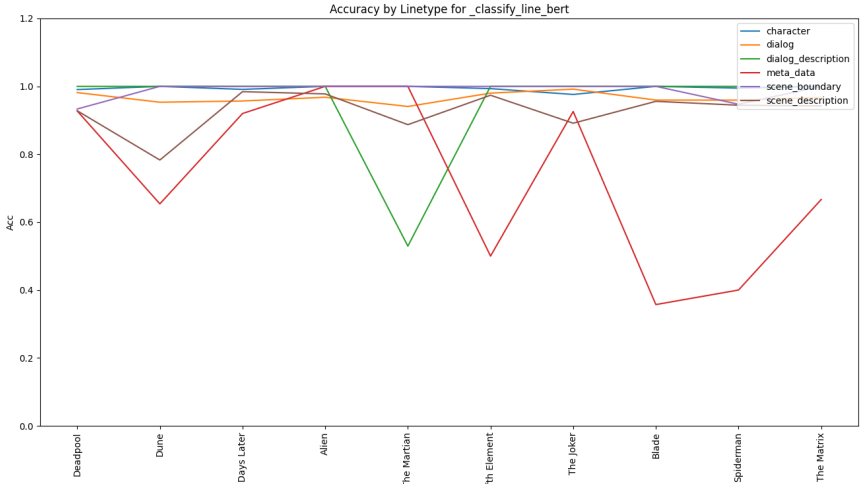

Figure 13: Accuracy by Linetype (BERT)

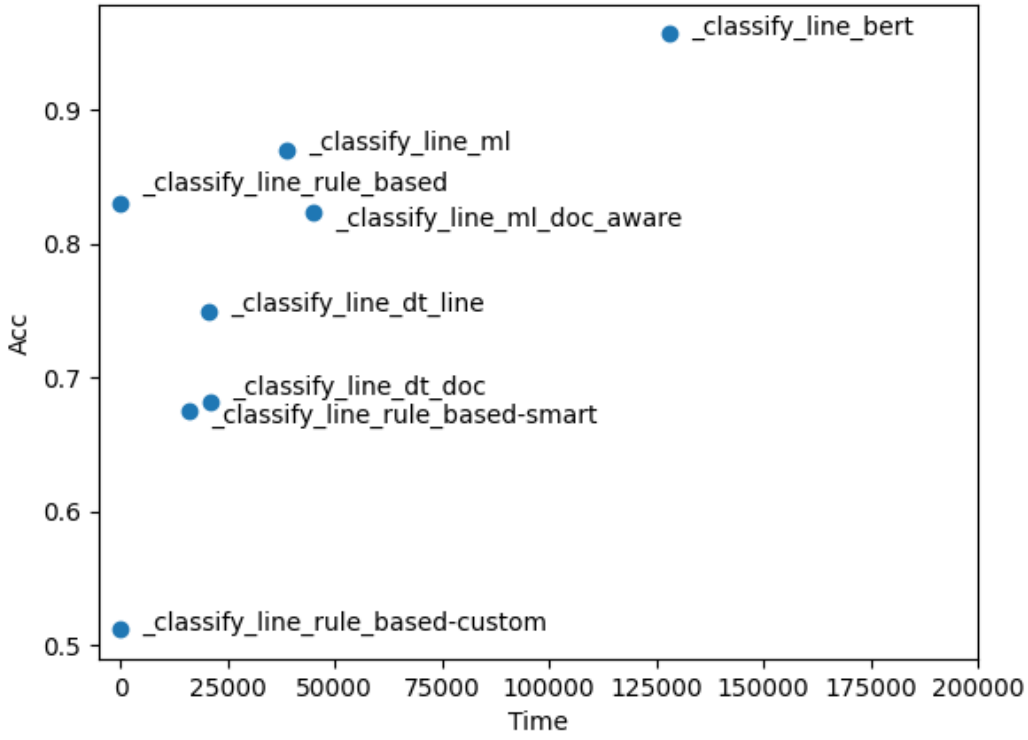

Figure 14: Parsing Time against Accuracy (in ms)