Improving Segmentation for Technical Support Problems

Kushal Chauhan [*]
ABV-IIITM, Gwalior
kushalchauhan98@gmail.com

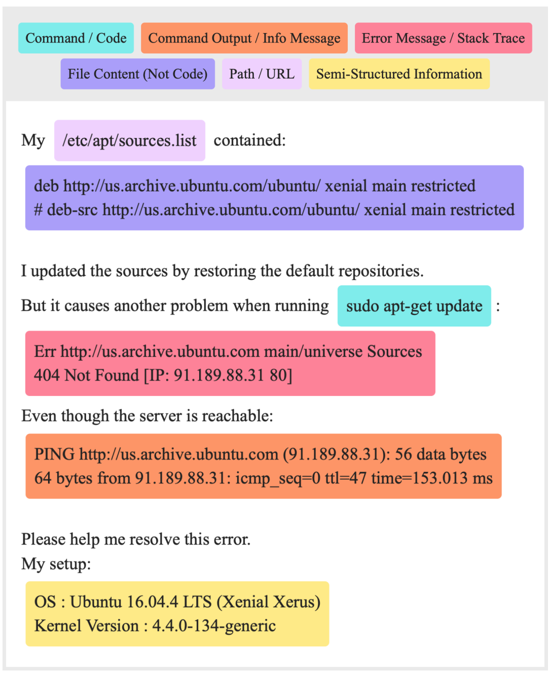
Abhirut Gupta [†]
IBM Research AI
abhirut.91@gmail.com

Abstract

Technical support problems are often long and complex. They typically contain user descriptions of the problem, the setup, and steps for attempted resolution. Often they also contain various non-natural language text elements like outputs of commands, snippets of code, error messages or stack traces. These elements contain potentially crucial information for problem resolution. However, they cannot be correctly parsed by tools designed for natural language. In this paper, we address the problem of segmentation for technical support questions. We formulate the problem as a sequence labelling task, and study the performance of state of the art approaches. We compare this against an intuitive contextual sentence-level classification baseline, and a state of the art supervised text-segmentation approach. We also introduce a novel component of combining contextual embeddings from multiple language models pre-trained on different data sources, which achieves a marked improvement over using embeddings from a single pre-trained language model. Finally, we also demonstrate the usefulness of such segmentation with improvements on the downstream task of answer retrieval.

1 Introduction

Problems, reported by users of software or hardware products - called *tickets* or *cases*, are often long and complex. Along with a description of the problem, users often report the setup, steps they have tried at mitigating the problem, and explicit requests. These problems also contain various non-natural language elements like snippets of code or commands tried, outputs of commands or software tools, error messages or stack traces, contents of log files or configuration files, and lists of key-value


Command / Code
Command Output / Info Message
Error Message / Stack Trace
File Content (Not Code)
Path / URL
Semi-Structured Information

My /etc/apt/sources.list contained:

```
deb http://us.archive.ubuntu.com/ubuntu/ xenial main restricted
# deb-src http://us.archive.ubuntu.com/ubuntu/ xenial main restricted
```

I updated the sources by restoring the default repositories.
But it causes another problem when running `sudo apt-get update` :

```
Err http://us.archive.ubuntu.com main/universe Sources
404 Not Found [IP: 91.189.88.31 80]
```

Even though the server is reachable:

```
PING http://us.archive.ubuntu.com (91.189.88.31): 56 data bytes
64 bytes from 91.189.88.31: icmp_seq=0 ttl=47 time=153.013 ms
```

Please help me resolve this error.
My setup:

```
OS : Ubuntu 16.04.4 LTS (Xenial Xerus)
Kernel Version : 4.4.0-134-generic
```

Figure 1: Various non-natural language segments labelled from a problem on AskUbuntu

pairs. Figure 1 shows a sample support problem from AskUbuntu [1], where all such segments are labeled.

While these segments are important sources of information for the human reader, they are difficult to handle for systems built to automatically answer support problems. As noted in Gupta et al. (2018), the non-natural language segments lead to parsing mistakes, and errors in the understanding of support problems. Correctly identifying these segments can also augment problem understanding. For instance, a retrieval engine with error messages and their solutions indexed in distinct fields would return better results with a fielded query containing just the error message from the ticket. Specialized tools for log analysis (He et al., 2016) could also be

[*] Work done at IBM Research during a summer internship
[†] Now at Google

[1] https://askubuntu.com/

run specifically on the identified log segment of problems.

In this paper, we aim to address the problem of identifying and extracting these non-natural language segments from support tickets. In particular, we choose to focus on the following six segment labels which appear often in support tickets (also shown in Figure 1):

- **Command / Code**: Includes terminal commands and programming code snippets
- **Command Output / Info Message**: Includes outputs of successful command/code executions
- **Error Message / Stack Trace**: Includes error traces resulting from unsuccessful command/code executions
- **File Content (Not Code)**: Includes contents of log files, configuration files, etc. which do not contain programming source code
- **Path / URL**: Includes file paths or webpage URLs
- **Semi-Structured Information**: Includes text which is structured in the form of key-value pairs, lists, etc., often used to convey system configurations or lists of components

We formulate the problem as a sequence labelling task, with word-level tags used to encode segments. To leverage the rich literature of supervised approaches in this framework, we also create a dataset with segments tagged for questions from AskUbuntu [2].

Our contributions are as follows -

1. We introduce a novel task towards understanding technical support problems, which has implications on a variety of downstream applications. We also release a tagged dataset of problems for the task.
2. We benchmark the performance of state of the art sequence labelling models on the task, studying their performance and limitations. This hopefully provides direction for future research.
3. Given the relatively small size of tagged data, we also explore pre-training based approaches. Our model leverages activations from multiple language models pre-trained on different data sources, and we show how they can be used to improve performance on the task.

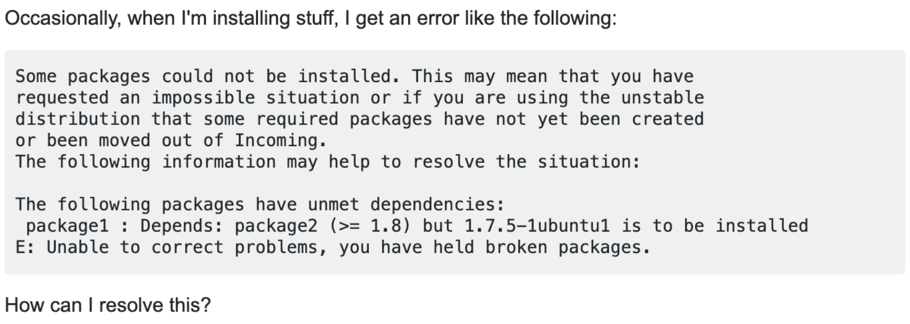
[2]Data available at https://github.com/kushalchauhan98/ticket-segmentation

2 Related Work

Understanding technical support problems is a particularly difficult task, owing to the long text of problems. In Gupta et al. (2018), the authors propose that understanding can be approached by extracting attributes of the ticket that correspond to the description of the problem (symptom), steps taken for mitigation (attempt), and explicit requests (intent). They also propose a dependency parser-based approach for extracting these attributes. However, while this approach pays attention to the semantics of the problem, the syntactical idiosyncrasies are ignored.

The idea of segmenting of questions for improvements on downstream tasks is not new. In Wang et al. (2010), the authors propose an unsupervised graph-based approach for segmenting questions from Community Question Answering (cQA) websites into sub-questions and their related context sentences. The authors demonstrate improvements in question retrieval by using these segments for more granular similarity matching. Chrupała (2013) uses representations from a character-level language model for segmenting code spans in Stack Overflow posts. The author uses $\langle code \rangle$ tags in HTML sources of posts for supervised training of a character level sequence labelling model. However, the $\langle code \rangle$ tags in the posts usually include all forms of non-natural language text like code snippets, command outputs, error messages or stack traces, and file paths (See Fig 2). The resulting level of granularity is thus insufficient for effective application in downstream tasks such as automated problem resolution. The task of text-segmentation in itself has been well studied in the literature, with popular unsupervised approaches like TextTiling (Hearst, 1997) and C99 (Choi, 2000). While, the problem of ticket segmentation, as defined by us, involves both segmenting and identifying segment types, we compare the performance of a more recent supervised segmentation approach (Koshorek et al., 2018) against our proposed model.

Significant amount of work has been done on us-

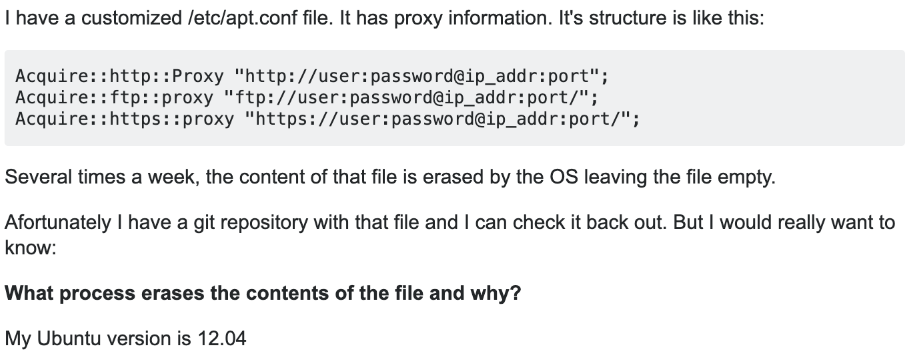

Occasionally, when I'm installing stuff, I get an error like the following:

```
Some packages could not be installed. This may mean that you have
requested an impossible situation or if you are using the unstable
distribution that some required packages have not yet been created
or been moved out of Incoming.
The following information may help to resolve the situation:

The following packages have unmet dependencies:
 package1 : Depends: package2 (>= 1.8) but 1.7.5-1ubuntu1 is to be installed
E: Unable to correct problems, you have held broken packages.
```

How can I resolve this?

(a)


I have a customized /etc/apt.conf file. It has proxy information. It's structure is like this:

```
Acquire::http::Proxy "http://user:password@ip_addr:port";
Acquire::ftp::proxy "ftp://user:password@ip_addr:port/";
Acquire::https::proxy "https://user:password@ip_addr:port/";
```

Several times a week, the content of that file is erased by the OS leaving the file empty.

Afortunately I have a git repository with that file and I can check it back out. But I would really want to know:

What process erases the contents of the file and why?

My Ubuntu version is 12.04

(b)

Figure 2: Sample problems from Ask Ubuntu with ⟨*code*⟩ tag used to present (a) an error message, and (b) contents of a configuration file

ing sequence labelling approaches for text segmentation tasks (Huang et al., 2015; Chiu and Nichols, 2016; Lample et al., 2016; Ma and Hovy, 2016; Rei et al., 2016; Peters et al., 2017). In Wang et al. (2018) the authors use ELMo embeddings and a biLSTM-CRF based architecture with self-attention for the task of neural discourse segmentation. We adopt a similar architecture, and explore the effect of using pre-trained contextual embeddings on our task. Given the fact that different segments in technical support problems have very different vocabularies, we also explore leveraging pre-trained Language Models on a variety of different datasets.

3 Data

Our dataset is derived from English questions on Ask Ubuntu. Questions posted on the website are similar to proprietary tech support tickets (in terms of question length, number of keywords/noun phrases, etc). We would like to point out that while posts on the website support the ⟨*code*⟩ HTML tag, it is not granular enough for our downstream tasks. These tags are also often abused to present snippets of command outputs/error messages/file paths etc. Figure 2 shows examples of such questions. We

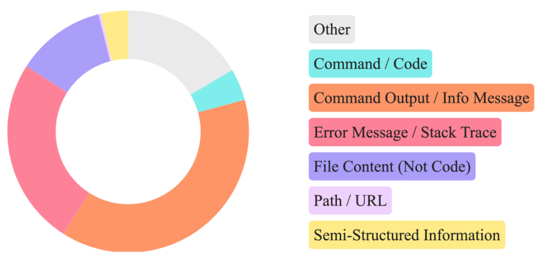


Figure 3: Relative frequencies of each tag in the dataset.

also do not use other metadata available (like turn-based information) with the data dump because these are not available with proprietary tickets.

Tagging is performed at the word level, and we use the BIO tagging scheme. We have a pair of Begin and Inside tags for each of the 6 non-natural language segments, and natural language segments are labelled O, totalling to 13 tags. We use the Doccano tool [3] for labelling, which provides better support for labelling long chunks in big documents compared to other popular sequence labelling annotation tools.

We obtain labelling for 1,317 questions, totalling

[3]https://github.com/chakki-works/doccano

	#Questions	**Avg. #Words**	**Avg. #Spans**						
			Total	**CC**	**CO**	**ES**	**FC**	**SS**	**PU**
Dataset	1317	897.37	4.86	2.13	1.20	0.62	0.30	0.14	0.46
Train	1053	899.33	4.91	2.14	1.20	0.63	0.30	0.14	0.49
Val	131	783.43	4.67	2.17	1.04	0.66	0.26	0.19	0.36
Test	133	994.10	4.64	2.08	1.36	0.47	0.35	0.09	0.28

Table 1: Statistics of the tagged dataset for segmentation with average number of words and spans per question. The last 6 columns contain average number of spans for each tag type - **CC**: Command/Code, **CO**: Command Output, **ES**: Error Message/Stack Trace, **FC**: File Content, **SS**: Semi-structured Information, **PU**: Path/URL

Annotator 2 / **Annotator 1**	**CC**	**CO**	**ES**	**FC**	**O**	**PU**	**SS**	**Total**	**Recall**
CC	928	382	18	8	29	0	0	1365	0.68
CO	10	7631	600	2351	67	0	316	10975	0.7
ES	29	586	7017	0	18	0	0	7650	0.92
FC	132	18	0	1984	1	0	211	2346	0.85
O	25	3	34	16	4995	6	0	5079	0.98
PU	0	0	1	0	23	89	0	113	0.79
SS	0	0	0	0	0	0	72	72	1
Total	1124	8620	7670	4359	5133	95	599	27600	1
Precision	0.83	0.88	0.92	0.46	0.97	0.94	0.12	1	

Figure 4: Confusion Matrix to show the word-level agreement between annotations of 2 annotators on 50 questions. The relatively large off-diagonal values represent the inherent difficulty in the task. Abbreviations for tags - **CC**: Command/Code, **CO**: Command Output, **ES**: Error Message/Stack Trace, **FC**: File Content, **SS**: Semi-structured Information, **PU**: Path/URL

to 11,580 spans (including spans labelled as O) and over 1.18 million words. We divide the data into 80:10:10 train, val, and test splits, at random. High-level statistics for the dataset are presented in Table 1. Figure 3 shows the average number of words per tag in the dataset. The tags *Command Output* and *Error Message* are relatively infrequent (1.2 and 0.6 per question) compared to the tag *Command Code* (2.1 per question), however, they cover a much larger fraction of words because they tend to be quite verbose.

In Figure 4 we show the inter-annotator agreement between two annotators on 50 questions. Few of the label pairs with large off-diagonal values include -

- *Command Output - Error Message*, which is understandable, as error messages are often interspersed in successful program runs. Conversely, unsuccessful program runs often contain a long train of success messages, only ending in one or few error logs.

- *Command Output - Semi-Structured Information* and *File Content - Semi-Structured Information*. This kind of confusion is due to the presence of network configurations, commands to view these, and files that contain these. They're often stored in configuration files as "key-value" pairs

- *Command Output - File Content*. This particular confusion stems from the "cat" command, and its use to view the contents of files.

The low inter-annotator agreement ($\kappa = 0.7637$) illustrates the inherent difficulty of the task. At this point, it's important to note that while there's some confusion in identifying labels for these segments, the need for these separate labels stems from downstream tasks.

4 Model

Given a technical support question, we formulate the segmentation problem as a sequence labelling task. It is an intuitive choice, given its efficacy for similar text segmentation problems like discourse segmentation (Wang et al., 2018) and chunking (Peters et al., 2017). Figure 5 presents an overview of our model. We explore different embeddings for each word (character-based embeddings, pre-trained embeddings, and pre-trained contextual embeddings). These word embeddings are then fed to a bi-directional GRU for encoding context. On the output of the GRU layer, we explore the effect of attention. Finally, the representations are passed to a CRF to decode the segment labels. We also study the impact of combining pre-trained contextual embeddings from multiple language models, trained on different data sources. In the rest of this section we detail individual components of the model.

4.1 Word Embeddings

For distributed word representations, we use skip-gram based word2vec embeddings (Mikolov et al., 2013) trained on all the questions from Ask Ubuntu.

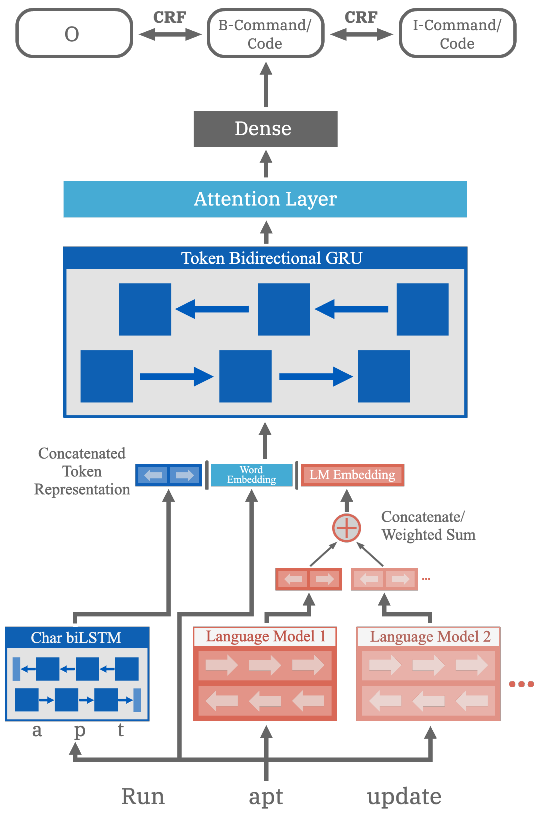


Figure 5: Model architecture for segmenting technical support problems.

We also look at fastText word embeddings (Bojanowski et al., 2017), which enrich word vectors by using subword information, emitting plausible word representations for unseen or rare words, giving us a significant gain. We use a 300-dimensional embedding from both word2vec and fastText.

4.2 Character Embeddings

In addition to the word-level features we also use bi-directional LSTM based character-level features similar to Chiu and Nichols (2016), Lample et al. (2016), and Ma and Hovy (2016). These features encode rich character level information which can improve performance, especially in syntactic tasks. We obtain an 80-dimensional representation for each word through the character bi-LSTM, which is the concatenation of the last hidden state of the forward and backward LSTMs.

4.3 Contextual Embeddings from Language Models

Pre-trained contextual embeddings have been shown to work well on a wide variety of NLP tasks. In domains with relatively small task-specific training data, the gains have been substantial (McCann et al., 2017; Akbik et al., 2018; Peters et al., 2017). We also include contextual embeddings from the pre-trained bi-directional language model in ELMo (Peters et al., 2018).

We observe that the non-natural language segments exhibit wide differences in syntactic and semantic structure, as is evident from Fig 1. We propose contextual embeddings from multiple language models; each trained on a different data source - English text, code snippets, config/log file contents. We hypothesize that combined embeddings from language models trained on separate data sources can capture word relationships better and can give richer word representations, as opposed to a single model trained on a large English corpora.

For combining multiple contextual embeddings, we explore two techniques - (1) a naive concatenation, and (2) a weighted sum, with weights learned from context-independent DME (Dynamic Meta-Embeddings) and context-dependent CDME (Contextualised Dynamic Meta-Embeddings) self-attention mechanisms as proposed by Kiela et al. (2018).

4.3.1 DME and CDME

When using embeddings from n different LMs for a training instance with s tokens $\{\mathbf{t}_j\}_{j=1}^{s}$, we get contextual embeddings $\{\mathbf{w}_{i,j}\}_{j=1}^{s} \in \mathbb{R}^{d_i}(i = 1, 2, \ldots, n)$.

For computing the weighted sum, the embeddings from multiple LMs are first projected to a common d'-dimensional space by learned linear functions:

$$\mathbf{w}'_{i,j} = \mathbf{P}_i\mathbf{w}_{i,j} + \mathbf{b}_i(i = 1, 2, \ldots, n) \quad (1)$$

where $\mathbf{P}_i \in \mathbb{R}^{d' \times d_i}$ and $\mathbf{b}_i \in \mathbb{R}^{d'}$. The projected embeddings are then combined with a weighted sum

$$\mathbf{w}_j^{DME} = \sum_{i=1}^{n} \alpha_{i,j}\mathbf{w}'_{i,j} \quad (2)$$

where $\alpha_{i,j} = g(\{\mathbf{w}'_{i,j}\}_{j=1}^{s})$ are scalar weights. In DME, they are learned with the self-attention mechanism:

$$\alpha_{i,j} = g\left(\mathbf{w}'_{i,j}\right) = \phi\left(\mathbf{a} \cdot \mathbf{w}'_{i,j} + b\right) \quad (3)$$

where $\mathbf{a} \in \mathbb{R}^{d'}$ and $b \in \mathbb{R}$ are learned parameters and ϕ is the softmax function.

For CDME, the self-attention mechanism is made context-dependent:

$$\alpha_{i,j} = g\left(\{\mathbf{w}'_{i,j}\}_{j=1}^{s}\right) = \phi\left(\mathbf{a} \cdot \mathbf{h}_j + b\right) \quad (4)$$

where $\mathbf{h}_j \in \mathbb{R}^{2m}$ is the j^{th} hidden state of a bi-directional LSTM which takes $\{\mathbf{w}'_{i,j}\}_{j=1}^{s}$ as input, $\mathbf{a} \in \mathbb{R}^{2m}$ and $b \in \mathbb{R}$. m is the number of hidden units in this LSTM, and it is set to 2 as in the original paper.

4.3.2 Data Sources for pre-trained LMs

In addition to the pre-trained ELMo model, we train three additional language models on different data sources. Each of these are also trained with the ELMo architecture. The pre-trained model emits word embeddings of size 1024, while each of our domain-specific models emit embeddings of size 256.

- **Code LM**: This LM was trained on a concatenation of all text inside the $\langle code \rangle$ tags of Ask Ubuntu, Super User, and Unix Stack Exchange posts. The total size of this corpus was approximately 400 MB.
- **Prog LM**: This LM was trained on a corpus containing programming source code that was compiled from various code repositories on GitHub. Approximately 275 MB in size, it includes sources in most popular languages such as C, C++, Python, Java, Go, JavaScript, and Bash.
- **Config LM**: This LM was trained on a corpus of configuration and log files present in the system folders of Mac OS and Ubuntu installations. The total size of the corpus was about 60 MB.

4.4 Attention

In Wang et al. (2018), the authors experiment with a restricted attention mechanism on top of the LSTM hidden representations. This is not appropriate for our task since the questions are fairly long (averaging around 900 words) and signals indicating the start or end of a segment might appear far away. Since RNNs are known to be poor at modelling very long-distance dependencies, we also experiment with the inclusion of the Scaled Dot-Product Attention layer (Vaswani et al., 2017) on top of the bi-directional GRU. This attention layer requires the computation of 3 matrices (Key, Query, Value) from the RNN hidden states, which entails a large number of extra parameters to be learned. Therefore, we also try a version of attention where all the three matrices are set equal to the hidden states of the GRU. We call these two approaches "weighted" and "un-weighted" attention, in our experiments.

5 Experimental Setup

With the setup above, we study the performance of various model components on the task of segmenting support problems. To put the performance in perspective, we also compare against three baselines detailed in Section 5.1. The evaluation metrics are carefully selected, avoiding an exact evaluation of such long and noisy segments, and rewarding partial retrieval of segments. The chosen evaluation metric is discussed in Section 5.2. Finally, to demonstrate the usefulness of the task, we evaluate the performance of answer retrieval with segmentation (Section 5.3).

All baselines and sequence labelling models are trained on the train split, and fine-tuned on the validation split. For the baselines, we only tune the regularization strength parameter. For the sequence labelling model, we tune the dropout and recurrent dropout parameters, as well as the learning rate. Our best performing models have a dropout of 0.3, recurrent dropout of 0, and learning rate of 1e-3. All results are then reported on the test split.

5.1 Baseline

The task of segmenting technical support problems can be thought to be comprised of two distinct subtasks - (1) segmentation of text, (2) identification of the segment label. With these in mind, we propose 3 baseline methods -

1. Sentence Only Baseline - Segmentation is done trivially with newlines and sentence boundaries serving as segment boundaries. The label for a segment is determined using just the current sentence as input.
2. Sentence Context Baseline - Segmentation is done identically to the Sentence Only baseline. The label for a segment is determined using the immediate neighbouring sentences along with the current sentence as input.
3. Supervised Text Segmentation Baseline - Segments are identified with the supervised algorithm for segmenting text as described in Koshorek et al. (2018). The label for each segment is identified with all the text contained in it as input.

For training the supervised text segmentation model from Koshorek et al. (2018) we use the whole data dump from AskUbuntu, with the $\langle code \rangle$ and $\langle /code \rangle$ html tags serving as segment boundaries.

For identifying segments (in all three baselines) we use a Logistic Regression classifier with representation from ELMo as input features. Segment representations are created by mean pooling the contextual representation of the comprising words from ELMo.

5.2 Evaluation Metrics

Segments in our dataset are typically quite long, therefore evaluation based on an exact match is quite harsh. Keeping this in mind, we resort to soft precision and recall metrics. We adopt proportional overlap based metrics, used for the task of opinion expression detection, as proposed by Johansson and Moschitti (2010).

Towards the calculation of soft precision and recall, consider two spans s and s' with labels l and l' respectively. The *span coverage*, c, is defined as how well s' is covered by s:

$$c\left(s, s'\right) = \frac{|s \cap s'|}{|s'|} \text{ if } l = l', 0 \text{ otherwise} \qquad (5)$$

Using span coverage, the *span set coverage* of a set of spans $\boldsymbol{S}$ with respect to another set of spans $\boldsymbol{S'}$ is computed as follows:

$$C\left(\boldsymbol{S}, \boldsymbol{S'}\right) = \sum_{s_j \in \boldsymbol{S}} \sum_{s'_k \in \boldsymbol{S'}} c\left(s_j, s'_k\right) \qquad (6)$$

Using the span set coverage, we can now define the soft precision P and recall R of a predicted set of spans $\hat{\boldsymbol{S}}$ with respect to the gold standard set of spans $\boldsymbol{S}$:

$$P(\boldsymbol{S}, \hat{\boldsymbol{S}}) = \frac{C(\boldsymbol{S}, \hat{\boldsymbol{S}})}{|\hat{\boldsymbol{S}}|} \quad R(\boldsymbol{S}, \hat{\boldsymbol{S}}) = \frac{C(\hat{\boldsymbol{S}}, \boldsymbol{S})}{|\boldsymbol{S}|} \qquad (7)$$

In this equation, the operator $| \cdot |$ counts the no. of spans in the span set.

5.3 Retrieval

An important task in the automation of technical support is the retrieval of the most relevant answer document for a given ticket (from a corpus of product documentation, FAQ docs, frequent procedures). In this experiment we demonstrate the usefulness of segmenting support tickets towards this goal. We index the text of about 250,000 answers from AskUbuntu with ElasticSearch [4]. Answers with a large number of downvotes, and very short answers are ignored. We use questions from our annotated dataset as search queries. We then compare the retrieval performance of querying with the whole question against a query with separate fields corresponding to each segment. In the fielded query, we set different boost values for the identified segments. Boosting a specific segment of the question with a higher value causes it to have more significance in the relevance score calculation in ElasticSearch. To decide the boost values, we calculate the average percentage word overlap between a segment in the question and its correct answer from AskUbuntu on the *train* and *val* sets. To compare retrieval performance, we evaluate the Mean Reciprocal Rank (MRR) of the correct answer for questions in the *test* set.

6 Results

Table 2 presents evaluation metrics for the three baselines against three variants of our sequence labelling model. The first variant does not use pre-trained embeddings from language models, the second uses just pre-trained ELMo, while the third combines pre-trained embeddings from multiple language models using CDME. All three variants use fastText for word embeddings (refer Section 6.1), character-based embeddings, and do not have attention mechanism before the final CRF layer (refer Section 6.2).

As one would expect, the Context Baseline performs much better than the Sentence Only Baseline. The sequence labelling models, however, outperform both the baselines by a huge margin, demonstrating the effectiveness of the model on the task. Specifically, the best performance is achieved by combining pre-trained embeddings from multiple language models trained on different data sources. It significantly outperforms the model using embeddings from a single pre-trained model on English (explored in Section 6.3).

In the following section we present results from the various model components we explored.

6.1 Effect of fastText

Row 1 and 4 in Table 3 presents the comparison between models using word embeddings from

[4] https://www.elastic.co/products/elasticsearch

Model	P	R	F1
Sent. Only Baseline	47.77	31.75	38.15
Sent. Context Baseline	52.52	34.03	41.3
Supervised Text Segmentation Baseline	44.13	40.43	42.20
SL w/o LM embeddings	74.57	75.51	75.04
SL + pre-trained ELMo	76.88	74.49	75.67
SL + CDME combined pre-trained Embeddings	78.30	79.29	**78.80**

Table 2: Results comparing the three baselines against variants of our sequence labelling model. The best performing variant uses CDME to combine pre-trained embeddings from multiple language models trained on different datasources.

Model	P	R	F1
Word2Vec (w/o Attn)	65.20	58.59	61.72
+ weighted Attn.	62.34	57.0	59.55
+ un-weighted Attn.	69.21	56.15	62.0
fastText	74.57	75.51	75.04

Table 3: Results for experiments between using Word2Vec and fastText embeddings. Also includes results of using attention on top of the model with Word2Vec. Since attention results were not promising, we did not repeat them with fastText.

word2vec and fastText. Both word2vec and fastText embeddings are trained on all posts in the Ask Ubuntu dataset. As we can see, fastText gives a marked improvement over using embeddings from word2vec. This is probably due to the nature of the vocabulary in our task. Since large portions of questions are spans of *command output* or *error messages* a lot of tokens appear very rarely. In fact, out of the 62,501 unique tokens in the dataset, 57% appear just once, and 78% appear 3 or fewer times. However, the characters in these tokens are probably very informative (for example "http" in a token would signal that the token is a URL). Therefore, fastText, which uses n-grams from a token to compute embeddings, would emit more meaningful representations.

As a simple experiment, we check the similarity of two URLs from the dataset that appear just once - *http://paste.ubuntu.com/1403448/* and *http://paste.ubuntu.com/14545476/*. While the cosine similarity of Word2Vec vectors for the two is -0.07, the similarity between the fastText vectors is 0.99.

6.2 Effect of Attention

Given the long tickets in our dataset, and unreasonably long lengths of spans for labels like *command output* or *error messages*, we explored the usefulness of attention in our model. We used the Scaled Dot-Product Attention as in (Vaswani et al., 2017). Rows 2 and 3 in Table 3 present the results of using attention. We find that weighted attention actually hurts performance. This could be because of the large number of extra parameters introduced in the calculation of Key, Value, and Query matrices. While the un-weighted version gets around this by using the bi-directional GRU hidden states as all 3 matrices, it doesn't improve results significantly either.

6.3 Effect of Contextual Pre-Trained Embeddings

As detailed in Section 4.3, we explore the impact of pre-trained contextual embeddings. We also test our hypothesis, that combining pre-trained embeddings from different data sources would perform better on our task than using embeddings from a language model trained on a single data source. The combination is also performed in two ways - naive concatenation of embeddings from all language models, and weighted combination using DME and CDME as in Kiela et al. (2018).

Table 4 summarizes these results. For the simple concatenation method, we present results for the best n-way combination of embeddings from different data sources, for each n (1, 2, 3, and 4). We find that combining embeddings from multiple language models trained on different data sources considerably outperforms using embeddings from a single pre-trained model (using both the naive concatenation and CDME). This is an artifact of the support problems containing large sections of non-natural language text. We also find that contextual weighting does better than a simple concatenation.

Model	P	R	F1
No Pretraining	74.57	75.51	75.04
Simple Concat - 1 (en)	76.88	74.49	75.67
Simple Concat - 2 (en + config)	77.67	76.12	76.89
Simple Concat - 3 (en + code + config)	79.64	77.72	78.67
Simple Concat - 4 (ALL)	76.05	76.65	76.35
DME	77.42	75.82	76.61
CDME	78.30	79.29	**78.80**

Table 4: Results comparing the models using various pre-trained embeddings. The *en* data source is the downloaded pre-trained ELMo model. For simple concatenation, we present the results for the best model at each n combinations of data sources. For example, when concatenating any 2 datasources, the *en + config* combination gives the best performance.

Method	MRR
Full Question	0.292
Segmented Question - Gold	0.300
Segmented Question - Predicted	**0.298**

Table 5: Retrieval results, comparing the performance of querying with the full question against segmented question (gold segments and predicted segments)

6.4 Retrieval of the Correct Answer

Table 5 presents results for the retrieval experiment. We show that weighing identified segments of the question with separate weights improves retrieval of the correct answer over a query with all tokens from the question. We also present results from the gold annotations of segments for these questions, as an upper-bound of the performance improvement we can hope to achieve.

7 Conclusion

In this paper, we introduce and address an important problem towards a better understanding of support tickets - segmentation of various non-natural language segments. We create an annotated dataset for the task, on questions from the publicly available website, Ask Ubuntu. We also study the performance of the most recent Recurrent Neural Network-based approaches to sequence labelling, on this task. In the end, we propose the novel idea of combining pre-trained embeddings from language models trained on different data sources, which substantially improves performance. We also demonstrate the usefulness of the task with improvements in retrieval of the correct answer. Our future research direction includes a thorough study of differences in this dataset with actual tickets, and potential for transfer. It is still valuable to study models on open datasets, however, as these are readily available to the community.

References

Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Jason P.C. Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics*, 4:357–370.

Freddy Y. Y. Choi. 2000. Advances in domain independent linear text segmentation. In *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference*, NAACL 2000, page 2633, USA. Association for Computational Linguistics.

Grzegorz Chrupała. 2013. Text segmentation with character-level text embeddings. Workshop on Deep Learning for Audio, Speech and Language Processing, ICML 2013, Atlanta, United States.

Abhirut Gupta, Anupama Ray, Gargi Dasgupta, Gautam Singh, Pooja Aggarwal, and Prateeti Mohapatra. 2018. Semantic parsing for technical support questions. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3251–3259, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2016. Experience report: system log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 207–218.

Marti A. Hearst. 1997. TextTiling: Segmenting text into multi-paragraph subtopic passages. *Computational Linguistics*, 23(1):3364.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF Models for Sequence Tagging. *arXiv e-prints*, page arXiv:1508.01991.

Richard Johansson and Alessandro Moschitti. 2010. Syntactic and semantic structure for opinion expression detection. In *Proceedings of the Fourteenth*

Conference on Computational Natural Language Learning, CoNLL 10, page 6776, USA. Association for Computational Linguistics.

Douwe Kiela, Changhan Wang, and Kyunghyun Cho. 2018. Dynamic meta-embeddings for improved sentence representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1466–1477, Brussels, Belgium. Association for Computational Linguistics.

Omri Koshorek, Adir Cohen, Noam Mor, Michael Rotman, and Jonathan Berant. 2018. Text segmentation as a supervised learning task. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 469–473, New Orleans, Louisiana. Association for Computational Linguistics.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, San Diego, California. Association for Computational Linguistics.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074, Berlin, Germany. Association for Computational Linguistics.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS17, page 62976308, Red Hook, NY, USA. Curran Associates Inc.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA. Curran Associates Inc.

Matthew Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. 2017. Semi-supervised sequence tagging with bidirectional language models. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1756–1765, Vancouver, Canada. Association for Computational Linguistics.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Marek Rei, Gamal Crichton, and Sampo Pyysalo. 2016. Attending to characters in neural sequence labeling models. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 309–318, Osaka, Japan. The COLING 2016 Organizing Committee.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Kai Wang, Zhao-Yan Ming, Xia Hu, and Tat-Seng Chua. 2010. Segmentation of multi-sentence questions: Towards effective question retrieval in CQA services. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR 10, page 387394, New York, NY, USA. Association for Computing Machinery.

Yizhong Wang, Sujian Li, and Jingfeng Yang. 2018. Toward fast and accurate neural discourse segmentation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 962–967, Brussels, Belgium. Association for Computational Linguistics.