

## INTRODUCTION

Text summarization is the process of generating short, fluent, and most importantly accurate summary of a respectively longer text document (Brownlee, 2017a). The main idea behind automatic text summarization is to be able to find a short subset of the most essential information from the entire set and present it in a human-readable format. As online textual data grows, automatic text summarization methods have potential to be very helpful because more useful information can be read in a short time.

Juniper Networks is a networking company that manufactures and supports enterprise-grade routing, switching and security products as well as service agreements (Juniper.net, 2018). In order to satisfy the customer base, Juniper tries to resolve issues quickly and efficiently. Juniper Networks maintains a Knowledge Base (KB) which is a dataset composed of questions from customers with human written solutions. The KB contains over twenty thousand articles. The company is currently developing a chatbot to provide 24x7 fast assistance on customer questions. The chatbot can search queries asked by the users in the KB and fetch links to the related articles. Juniper Networks is looking for ways to be able to automatically summarize these articles so that chatbot can present the summaries to the customers. The customers can then decide if they would like to read the entire article. The summarization tool could be further used internally for summarizing tickets and issues created by Juniper's employees.

The goals of this Major Qualifying Project are to research methods for text summarization, create an end-to-end prototype tool for summarizing documents and identify if Juniper Networks' datasets can be summarized effectively and efficiently. In order to achieve these goals, we developed the following objectives:

- Research current technologies and progress associated with text summarization.

- Filter and clean datasets to be used for summarization.
- Implement algorithms and models for different methods of text summarization.
- Evaluate the models and tune them if necessary.
- Build and host an end-to-end tool which takes texts as input and outputs a summary

Following the above objectives, we investigated extractive summarization and abstractive summarization, commonly used text summarization methods. We implemented extractive summarization using Text rank (Mihalcea, Rada, and Paul Tarau, 2004) and TF-IDF algorithms (Ramos and Juan, 2003). We implemented abstractive summarization using deep learning models. To run and test our implementations, we chose and filtered five datasets (three Juniper datasets and two public datasets). Using the cleaned datasets, we trained and evaluated different versions of our model. Once we finalized the model, we built an end-to-end tool web application that can summarize any given input text. The web application offers choices for summarizing input text from each of the five datasets: the News dataset, the Stack Overflow dataset, Juniper's KB dataset, Juniper's JIRA dataset and Juniper's JTAC dataset.

The rest of this report is organized as follows. Section 2 discusses the technical terms related to text summarization. Next, Section 3 introduces some related works on text summarization. Lastly, Section 4 and 5 present the methods used to achieve the project goal and the results.

## Background

This section explores the technologies which were used in this project (Section 2.1 - 2.3). The section first discusses the key concepts for text summarization, followed by the metrics used to evaluate them along with the environments (Section 2.4) and the libraries used to complete this project (Section 2.5).

### Natural Language Processing

Natural Language Processing (NLP) is a field in Computer Science that focuses on the study of the interaction between human languages and computers (Chowdhury, 2003). Text summarization is in this field because computers are required to understand what humans have written and produce human-readable outputs. NLP can also be seen as a study of Artificial Intelligence (AI). Therefore, many existing AI algorithms and methods, including neural network models, are also used for solving NLP related problems. With the existing research, researchers generally rely on two types of approaches for text summarization: extractive summarization and abstractive summarization (Dalal and Malik, 2013).

### Text Extraction

Extractive summarization means extracting keywords or key sentences from the original document without changing the sentences. Then, these extracted sentences can be used to form a summary of the document.

### Text rank

Text rank is an algorithm inspired by Google's PageRank algorithm that helps identify key sentences from a passage (Mihalcea, Rada, and Paul Tarau, 2004). The idea behind this

algorithm is that the sentence that is similar to most other sentences in the passage is probably the most important sentence in the passage. Using this idea, one can create a graph of sentences connected with all the similar sentences and run Google's PageRank algorithm on it to find the most important sentences. These sentences would then be used to create the summary.

## TF-IDF

Term Frequency-Inverse Document Frequency (TF-IDF) is used to determine the relevance of a word in the document (Ramos and Juan, 2003). The underlying algorithm calculates the frequency of the word in the document (term frequency) and multiplies it by the logarithmic function of the number of documents containing that word over the total number of documents in the dataset (inverse document frequency). Using the relevance of each word, one can compute the relevance of each sentence. Assuming that most relevant sentences are the most important sentences, these sentences can then be used to form a summary of the document.

## Text Abstraction

Compared to extractive summarization, abstractive summarization is closer to what humans usually expect from text summarization. The process is to understand the original document and rephrase the document to a shorter text while capturing the key points (Dalal and Malik, 2013). Text abstraction is primarily done using the concept of artificial neural networks. This section introduces the key concepts needed to understand the models developed for text abstraction.

## Artificial Neural Network

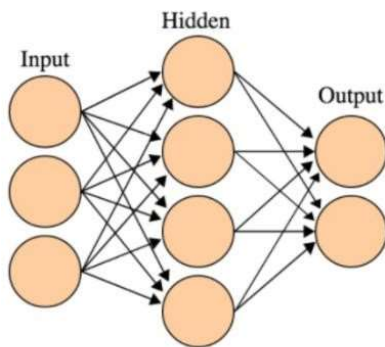


Figure 1: Simple neural network model

(Cburnett, 2016)

Artificial neural networks are computing systems inspired by biological neural networks. Such systems learn tasks by considering examples and usually without any prior knowledge. For example, in an email spam detector, each email in the dataset is manually labeled as “spam” or “not spam”. By processing this dataset, the artificial neural networks evolve their own set of relevant characteristics between the emails and whether a new email is spam.

To expand more, artificial neural networks are composed of artificial neurons called units usually arranged in a series of layers. Figure 1 is the most common architecture of a neural network model. It contains three types of layers: the input layer contains units which receive inputs normally in the format of numbers; the output layer contains units that “respond to the input information about how it is learned any task”; the hidden layer contains units between input layer and output layer, and its job is to transform the inputs to something that output layer can use (Schalkoff, 1997).

## RNN and LSTM

Traditional neural networks do not recall any previous work when building the understanding of the task from the given examples. However, for tasks like text summarization, the sequence of words in input documents is critical. In this case, we want the model to remember the previous words when it processes the next one. To be able to achieve that, we have

to use recurrent neural networks because they are networks with loops in them where information can persist in the model (Christopher, 2015).

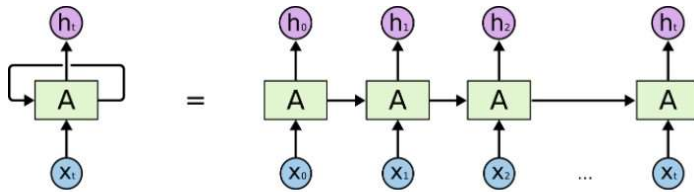


Figure 2: An unrolled recurrent neural network.  
(Christopher, 2015)

Figure 2 shows how an Recurrent neural network (RNN) looks like if it is unrolled. For the symbols in the figure, “ $h_t$ ” represents the output units value after each timestamp (if the input is

a list of strings, each timestamp can be the processing of one word), “ $x$ ” represents the input units for each timestamp, and  $A$  means a chunk of the neural network. Figure 2 shows that the result from the previous timestamp is passed to the next step for part of the calculation that happens in a chunk of the neural network. Therefore, the information gets captured from the previous timestamp. However, in practice,

traditional RNNs often do not memorize information efficiently with the increasing distance between the connected information. Since each activation function is nonlinear, it is hard to trace back to hundreds or thousands of operations to get the information.

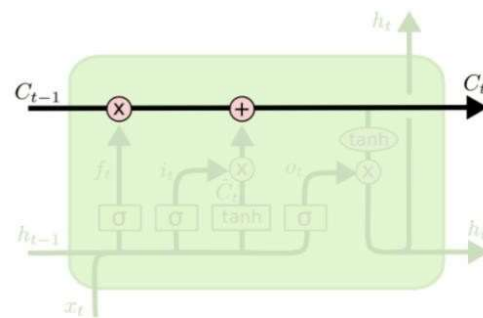


Figure 3: The linear operations in an LSTM cell  
(Christopher, 2015)

Fortunately, Long Short-Term Memory (LSTM) networks can convey information in the long term. Different from the traditional RNN, inside each LSTM cell, there are several simple linear operations which allow data to be conveyed without doing the complex computation. As

shown in Figure 3, the previous cell state containing all the information so far smoothly goes through an LSTM cell by doing some linear operations.

Inside, each LSTM cell makes decisions about what information to keep, and when to allow reads, writes and erasures of information via three gates that open and close.

As shown in Figure 4, the first gate is called the “forget gate layer”, which takes the previous output units value  $h_{t-1}$  and the current input  $x_t$ , and outputs a number between 0 and 1 to indicate the ratio of passing information. 0 means do not let any information pass, while 1 means let all information pass.

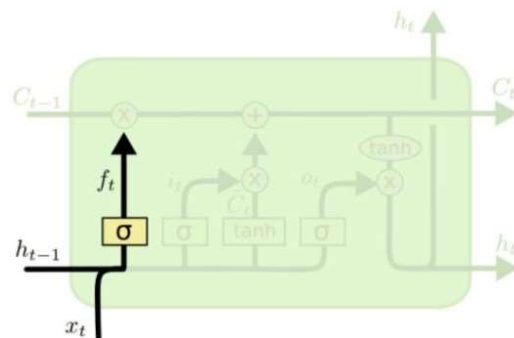


Figure 4: The forget gate layer  
(Christopher, 2015)

To decide what information needs to be updated, LSTM contains the “input gate layer”. It also takes in the previous output units value  $h_{t-1}$  and the current input  $x_t$  and outputs a number to indicate inside which cells the information should be updated. Then, the previous cell state  $C_{t-1}$  is updated to the new state  $C_t$ .

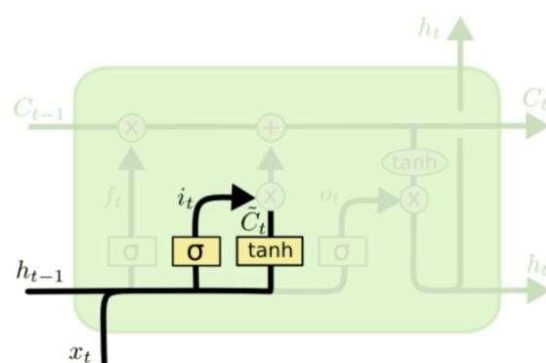


Figure 5: The input gate layer  
(Christopher, 2015)

The last gate is “output gate layer”,

going through a tanh function, and then it is multiplied by the weighted output of the sigmoid function. So, the output units value  $h_t$  is passed to the next LSTM cell (Christopher, 2015).

Simple linear operators connect the three gate layers. The vast LSTM neural network consists of many LSTM cells, and all information is passed through all the cells while the critical information is kept to the end, no matter how many cells the network has.

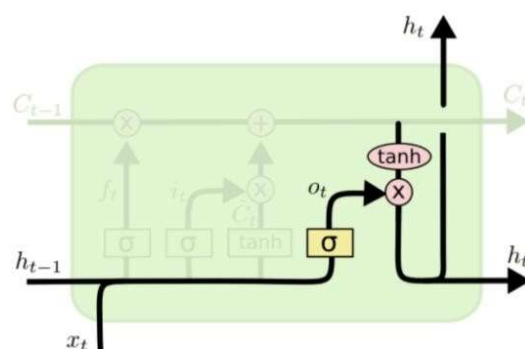


Figure 6: The output gate layer  
(Christopher, 2015)

## Word Embedding

Word embedding is a set of feature learning techniques in NLP where words are mapped to vectors of real numbers. It allows similar words to have similar representation, so it builds a relationship between words and allows calculations among them (Mikolov, Sutskeve, Chen, Corrado, and Dean, 2013). A typical example is that after representing words to vectors, the function “king - men + women” would ideally give the vector representation for the word “queen”. The benefit of using word embedding is that it captures more meaning of the word and often improves the task performance, primarily when working with natural language processing.

## ROUGE-N and BLEU Metrics

ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation. It is a set of metrics that is used to score a machine-generated summary using one or more reference summaries created by humans. ROUGE-N is the evaluation of N-grams recall over all the reference summaries. The recall is calculated by dividing the number of overlapping words over



the total number of words in the reference summary (Lin, Chin-Yew, 2004). The BLEU metric, contrary to ROUGE, is based on N-grams precision. It refers to the percentage of the words in the machine generated summary overlapping with the reference summaries (Papineni, Kishore, et al., 2002). For instance, if the reference summary is “There is a cat and a tall dog” and the generated summary is “There is a tall dog”, the ROUGE-1 score will be 5/8 and the BLEU score will be 5/5. This is because the number of overlapping words are 5 and the number of words in system summary and reference summary are 5 and 8 respectively. These two metrics are the most commonly used metrics when working with text summarization.

## Libraries

### Keras

Keras is a Python library initially released in 2015, which is commonly used for machine learning. Keras contains many implemented activation functions, optimizers, layers, etc. So, it enables building neural networks conveniently and fast. Keras was developed and maintained by François Chollet, and it is compatible with Python 2.7-3.6 (Keras.io, n.d.).

### NLTK

Natural Language Toolkit (NLTK) is a text processing library that is widely used in Natural Language Processing (NLP). It supports the high-performance functions of tokenization, parsing, classification, etc. The NLTK team initially released it in 2001 (Nltk.org, 2018).

### Scikit-learn

Scikit-learn is a machine learning library in Python. It performs easy-to-use dimensional reduction methods such as Principal Component Analysis (PCA), clustering methods such as k-

means, regression algorithms such as logistic regression, and classification algorithms such as random forests (Scikit-learn.org, 2018).

### **Pandas**

Pandas provides a flexible platform for handling data in a data frame. It contains many open-source data analysis tools written in Python, such as the methods to check missing data, merge data frames, and reshape data structure, etc. (“Pandas”, n.d.).

### **Gensim**

Gensim is a Python library that achieves the topic modeling. It can process a raw text data and discover the semantic structure of input text data by using some efficient algorithms, such as Tf-Idf and Latent Dirichlet Allocation (Rehurek, 2009).

### **Flask**

Flask, issued in mid-2010 and developed by Armin Ronacher, is a robust web framework for Python. Flask provides libraries and tools to build primarily simple and small web applications with one or two functions (Das., 2017).

### **Bootstrap**

Bootstrap is an open-source JavaScript and CSS framework that can be used as a basis to develop web applications. Bootstrap has a collection of CSS classes that can be directly used to create effects and actions for web elements. Twitter’s team developed it in 2011 (“Introduction”, n.d.).

### **GloVe**

Global Vectors for Word Representation (GloVe), which is initially developed by a group of Stanford students in 2014, is a distributed word representations model that performs better

than other models on word similarity, word analogy, and named entity recognition. (Pennington, Richard and Christopher, 2014).

## LXML

The XML toolkit IXML, which is a Python API, is bound to the C libraries libXML2 and libxslt. LXML can parse XML files faster than the Element Tree API, and it also derives the completeness of XML features from libXML2 and libxslt libraries (LXML.de, 2017)

### 3.0 Related Work

In order to build the text summarization tool for Juniper Networks, we first researched existing ways of doing text summarization. Text summarization, still at its early stage, is a field in Natural Language Processing (NLP). Deep learning, an area in machine learning, has performed with state-of-the-art results for common NLP tasks such as Named Entity Recognition (NER), Part of Speech (POS) tagging or sentiment analysis (Socher, R., Bengio, Y., & Manning, C., 2013). In case of text summarization, the two common approaches are extractive and abstractive summarization.

For extractive summarization, dominant techniques including TF-IDF and Text rank (Hasan, Kazi Saidul, & Vincent Ng, 2010). Text rank was first introduced by Mihalcea, Rada, and Paul Tarau in their paper *Text rank: Bringing order to text* (2004). The paper proposed the idea of using a graph-based algorithm similar to Google's Pagerank to find the most important sentences. Juan Ramos proposed TF-IDF (2003). He explored the idea of using a word's uniqueness to perform keyword extraction. This kind of extraction can also be applied to an entire sentence by calculating the TF-IDF of each word in the sentence. We implemented both of these algorithms - Text rank and TF-IDF, and compared their performances in different datasets.

Abstractive summarization is most commonly performed with deep learning models. One such model that has been gaining popularity is sequence to sequence model (Nallapati, Zhou, Santos, Gulçehre, & Xiang 2016). Sequence to sequence models have been successful in speech recognition and machine translation (Sutskever, I., Vinyals, O., & Le, Q. V., 2014). Recent studies on abstractive summarization have shown that sequence to sequence models using encoders and decoders beat other traditional ways of summarizing text. The encoder part encodes

the input document to a fixed-length vector. Then the decoder part takes the fixed-length vector and decodes it to the expected output (Bahdanau, Cho, & Bengio, 2014).

We focused on three recent pieces of research on text summarization as inspirations for our model of abstractive summarization (Rush, Chopra, & Weston, 2015; Nallapati, Zhou, Santos, Gulçehre, & Xiang 2016; Lopyrev, 2015). All three journals have used encoder-decoder models to perform abstractive summarization on the dataset of news articles to predict the headlines.

The model created by Rush et al., a group from Facebook AI Research, has used convolutional network model for encoder, and a feedforward neural network model for decoder (for details, please see Appendix A: Extended Technical Terms). In their model, only the first sentence of each article content is used to generate the headline (2015).

The model generated by Nallapati et al., a team from IBM Watson, used Long Short-Term Memory (LSTM) in both encoder and decoder. They used the same news article dataset as the one that the Facebook AI Research group used. In addition, the IBM Watson group used the first two to five sentences of the articles' content to generate the headline (2016). Nallapati et al. were able to outperform Rush et al.'s models in particular datasets.

The article from Konstantin Lopyrev talks about a model that uses four LSTM layers and attention mechanism, a mechanism that helps improve encoder-decoder model's performance (2015). Lopyrev also used the dataset of news articles, and the model predicts the headlines of the articles from the first paragraph of each article.

All three works show that encoder-decoder model is a potential solution for text summarization. Using LSTM layers in encoder-decoder also allow capturing more information from original article content than traditional RNNs. In this project, inspired by previous works,

we also used the encoder-decoder model with LSTM but in a slightly different structure. We used three LSTM layers in the encoder and another three LSTM layers in the decoder (details of the model are described in Section 3.0). However, the datasets used in this project were not as clean as news articles. Our datasets contain a lot of technical terms, coding languages as well as unreadable characters. Therefore, we tried to combine the extractive summarization and abstractive summarization to test if it provides better performance. We hoped that extractive summarization could help extract key sentences from the articles, which can be used as inputs to our abstractive deep learning models. This way, the input documents for the abstractive summarization would be neater than the original ones.

## Methodology

The goal of this project is to explore automatic text summarization and analyze its applications on Juniper's datasets. To achieve the goal, we completed the following steps:

Step 1: Choose and clean datasets

Step 2: Build the extractive summarization model

Step 3: Build the abstractive summarization model

Step 4: Test and compare models on different datasets

Step 5: Tune the abstractive summarization model

Step 6: Build an end-to-end application

## Choose and clean datasets

Section 4.1.1 introduces the basic information about each dataset we used, precisely the contents of the dataset, and the reason for using the dataset. Section 4.1.2 and 4.1.3 dive into the steps of data cleaning and categorizing.

## Datasets Information

We worked on five datasets — the Stack Overflow dataset (Stack Dataset), the news articles dataset (News Dataset), the Juniper Knowledge Base dataset (KB Dataset), the Juniper Technical Assistance Center Dataset (JTAC Dataset) and the JIRA Dataset. Each dataset consists of many cases, where each case consists of an article and a summary or a title. Since the raw News Dataset was already cleaned, we primarily focused on cleaning the rest four datasets. Figure 7 below shows the changes in dataset sizes before and after cleaning the data. As shown in the figure, after cleaning the datasets, we had two large datasets (the Stack Dataset and the KB

Dataset) with over 15,000 cases and two small datasets (the JTAC Dataset and the JIRA Dataset) with nearly 5,000 cases to work with.

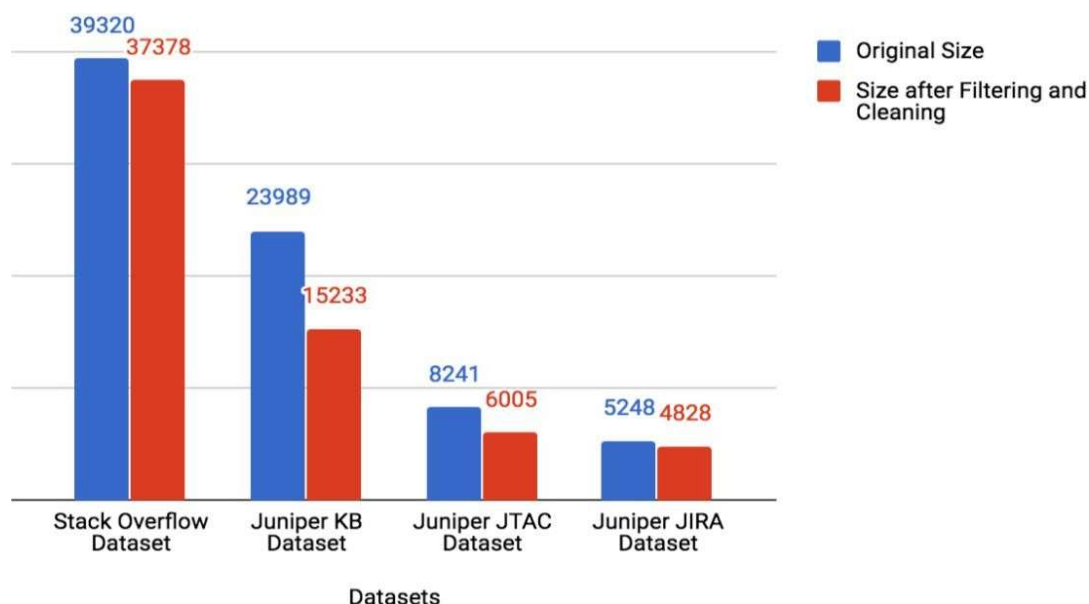


Figure 7: Dataset Information

The Stack Dataset is a collection of questions and answers on the StackOverflow website (stackoverflow.com, 2018). We used a filtered version of the Stack Dataset dealing only with networking related issues. There are 39,320 cases in this data frame, which is the largest dataset we worked on. For each case, we filtered the dataset to only keep the unique question id, the question title, the question body, and the answer body. Then we cleaned the filtered dataset by removing chunks of code, non-English articles and short articles. Finally, we got 37,378 cases after cleaning. The reason we chose to work with the Stack Dataset is because it contains technical questions similar to that of the KB Dataset. However, the Stack Dataset is supposedly cleaner than the KB Dataset, and by running our models in a cleaner dataset, we could first focus on designing our model to set a benchmark.



Second, the News Dataset is a public dataset containing news articles from Indian news websites. We used this dataset because the dataset includes a human-generated summary for each article, which can be used to train our model. For our purposes, we only used the article body and the summary of each article. This dataset was used just for extractive summarization as the dataset was not relevant to the Juniper KB Dataset.

Third, the KB Dataset, which is the one we put the most emphasis on, contains technical questions and answers about networking issues. The raw dataset is in a directory tree of 23,989 XML files, and each XML file contains the information about one KB article. For our training and testing, we only kept a unique document id, a title, a list of categories that the article belongs to, and a solution body for each KB article in the data frame. We filtered out the top 30 categories which contained 15,233 cases. Our goal was to use the KB articles' solutions as input and predict the KB articles' titles.

Fourth, the JTAC Dataset contains information about JTAC cases. It has 8,241 cases, and each case has a unique id, a synopsis, and a description. The raw dataset is in a noisy JSON file.

At last, the JIRA Dataset is about JIRA bugs from various projects. JIRA is a public project management tool developed by Atlassian for issue tracking. The JIRA Dataset has 5,248 cases, and each case has a unique id, a summary, and a description. Same as the JTAC Dataset, the raw JIRA Dataset is also in a JSON file.

## Data Cleaning

The five datasets we worked were very noisy containing snippets of code, invalid characters, and unreadable sentences. For an efficient training, our models needed datasets with no missing value and no noisy words. Based on this guideline, we followed these basic steps to clean our datasets:

- Read data file and make a data frame

The Stack Data are in CSV files and can be easily transferred to a data frame by using pandas library. However, the KB Dataset is stored in a directory tree of XML files, so we used Lxml to read each XML file from the root to each element and store the information we need into a data frame (LXML.de, 2017).

- Check for missing values.

Since fewer than 5% of the articles had missing values, the articles containing missing values were dropped from the data frames.

- Detect and remove the code part in all texts.

In the Stack Dataset and the KB Dataset, there are many chunks of code in the question and answer bodies. The code snippets would cause problems as the summarization models cannot capture the inference of the code. Therefore, we identified the chunks of code by locating the code tags in the input strings and deleting everything between “<code>” and “</code>” tags. Eventually, we found that nearly 33% of KB articles contain some type of code in them.

- Detect and remove the unknown words with “&” symbol in all texts.

In the KB Dataset, we found that there are 48.76% words which cannot be recognized by Juniper’s word embedding. Some of the unrecognized words are proper nouns, but some of them are garbled and meaningless words that start with “&” symbols such as the word “&npma”. The proper nouns might catch the unique information in the articles, so we did not remove them. However, we detected and removed all the unknown words started with “&” symbol.

- Detect and remove the Spanish articles.

In the KB Dataset, 164 articles are written in Spanish. Our project’s focus was only on English words, and having words outside of English language would cause problems while

training our models. We identified the Spanish articles by looking for some common Spanish words such as “de”, “la” and “los”. Any article containing a Spanish word was removed.

- Detect and remove the articles less than 10 characters or 3 words.

In the KB Dataset, some solution articles only contained a link or a period, so we removed any article less than 10 characters or 3 words.

- Detect and remove the articles that have many digits.

In the JTAC dataset, there are nearly 19% articles in which more than 20% of all characters are digits, and most of the digits are meaningless in the context such as “000x”. Digits are seldom used in training and may affect the prediction, so we removed such articles.

- Check duplicate articles and write the cleaned data into a CSV file.

We also checked whether there were duplicated data, and we found that all data are unique. Finally, we wrote the cleaned data into a comma-separated values file.

## **Data Categorization**

Data Categorization in our project involved categorizing KB articles by creating a hierarchy of the existing KB categories. Each KB article was associated with a list of categories like- “MX240”, “MX230”, “MX”, etc. In this case, the hierarchy of the categories should reflect category “MX240” to be a child node of category “MX”. “MX” is the name of a product series at Juniper, while “MX240” is the name of a product in the MX series. The goal of categorizing KB data is to have a more precise structure of the KB dataset which could be used by Juniper Networks for future data related projects as well. The detailed steps of categorizing KB articles are listed below:

- Get the set of all categories.

- Remove digits and underscores in each category name.

In order to efficiently categorize the data, we removed the digits and underscores at the beginning and the end of each category name. For example, “MX240\_1” is shrunk as “MX”. This was helpful when we used the Longest Common Substrings to categorize the data because the longest common substring among “MX240\_1” and “MX240\_2” is “MX240\_”, whereas ideally, we wanted the category to be just “MX”.

- Find a list of Longest Common Substrings (LCS) among category names.

Since similar category names are listed consecutively, we went through the entire set and found the LCS with minimum two characters among the neighbors. If a specific string did not have a common substring with its previous string and its successive string, this string was regarded as a parent node which had no child node.

- Manually examine the LCS and pick 30 decent category names.

After we listed all the common substrings, we manually took a look at the list and picked 30 category names which were meaningful and contained many children nodes. For example, we picked some names of main product series at Juniper such as “MX” and “EX”, and we also chose some networking-related categories such as “SERVER” and “VPN”.

- Write all KB articles that belonged to those 30 categories into a CSV file. Build the hierarchy map for KB categories.

The last step was to extract the KB articles that contained any category name that belonged to the target 30 categories in the category list. We also generated the hierarchy map for all KB categories, so it is easier for future category-cased extractions.

### Extractive Summarization

We began the text summarization by exploring the extractive summarization. The goal was to try the extractive approach first and use the output from extraction as an input of the abstractive summarization. After experimenting with the two approaches, we would then pick the best approach for Juniper Network's Knowledge Base (KB) dataset. The text extraction algorithms and controls were implemented in Python. The code contained three important components - the two algorithms used, the two control methods, and the metrics used to compare all the results.

### Algorithms

The algorithms used for text extraction were - Textrank (Mihalcea, Rada, and Paul Tarau, 2004) and TF-IDF (Ramos and Juan, 2003). These two algorithms were run on three datasets - the News Dataset, the Stack Dataset and the KB Dataset. Each algorithm generates a list of sentences in the order of their importance. Out of that list, the top three sentences were joined together to form the summary.

Textrank was implemented by creating a graph where each sentence formed the node, and the edges were the similarity score between the two node sentences. The similarity score was calculated using Google's Word2Vec public model. The model provides a vector representation of each word which can be used to compute the cosine similarity between them. Once the graph was formed, Google's PageRank algorithm was executed in the graph, and then top sentences were collected from the output.

Scikit Learn's TF-IDF module was used to compute the TF-IDF score of each word with respect to its dataset. Each sentence was scored by calculating the sum of the scores of each word

in the sentences. The idea behind this was that the most important sentence in the document is the sentence with the most uniqueness (most unique words).

## Control Experiment

To be able to verify the effectiveness of the two algorithms, two control experiments were used. A control experiment is usually a naive procedure that helps in testing the results of an experiment. The two control experiments used in text extraction were - forming a summary by combining the first three lines and forming a summary by combining three random lines in the article. By running the same metrics in these control experiments as the experiments with the algorithm being tested, a baseline can be created for the algorithms. In an ideal case, the performance of the algorithms should always be better than the performance of the control experiments.

## Metrics

All the extracted results including the results from the control experiments were evaluated using the ROGUE-1 score and the BLEU score. The score for each of the generated summary was computed. Then, each dataset was scored by computing the mean of the summary scores in the dataset. These scores were then compared and the best algorithm for each dataset was picked.

## Abstractive Summarization

The next step in our project was to work with abstractive summarization. This is one of the most critical components of our project. We used deep learning neural network models to create an application that could summarize a given text. The goal was to create and train a model which can take sentences as the input and produce a summary as the output.

The model would have pre-trained weights and a list of vocabulary that it would be able to output. Figure 8 shows the basic flow of our data we wanted our model to achieve. The first step was to convert each word to its index form. In the figure, the sentences “I have a dog. My dog is tall.” is converted to its index form by using a word to index map.

The index form was then passed through an embedding layer which in turn converted the indexes to vectors. We used pre-trained word embedding matrixes to achieve this. The output from the embedding layer was then sent as an input to the model. The model would then compute and create a one-hot vectors matrix of the summary. A one-hot vector is a vector of dimension equal to the size of the model’s vocabulary where each index represents the probability of the output to be the word in that index of the vocabulary. For example, if the index 2 in the one-hot vector is 0.7, the probability of the result to be the word in the vocabulary index 2 is 0.7. This matrix would then be converted to words by using the index to word mapping that was created using the word to index map. In the figure, the final one-hot encoding when converted to words forms the expected summary “I have a tall dog.”. Section 4.3.1, 4.3.2 and 4.3.3 expand our architecture of the above model in detail.

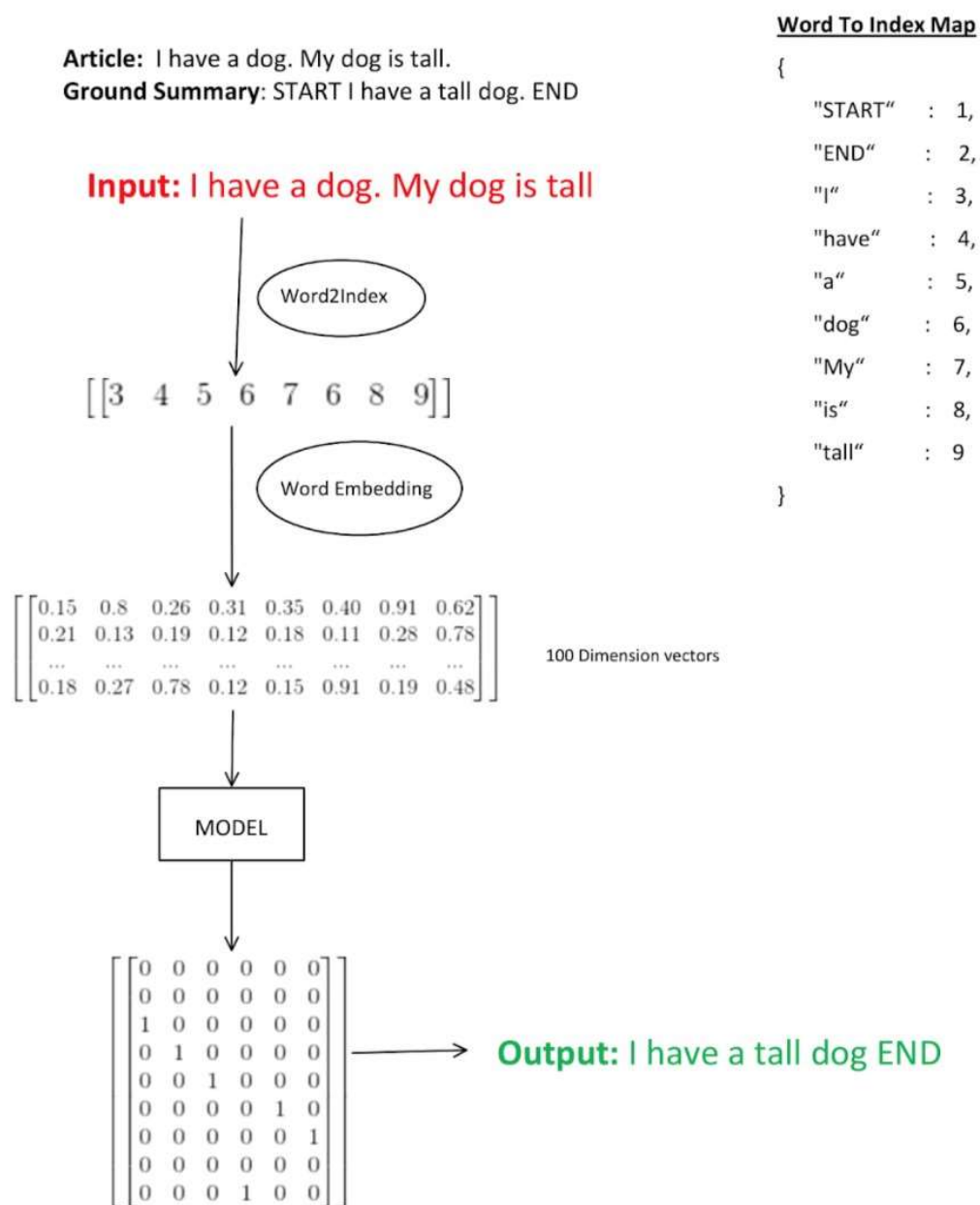


Figure 8: Abstractive summarization basic flow diagram

## Preparing the dataset

Before training the model on the dataset, certain features of the dataset were extracted. These features were used later when feeding the input data to the model and comprehending the output information from the model.



We first collected all the unique words from the input (the article) and the expected output (the title) of the documents and created two Python dictionaries of vocabulary with index mappings for both the input and the output of the documents.

In addition, we created an embedding matrix by converting all the words in the vocabulary to its vector form using pre-trained word embeddings. For public datasets like Stack Overflow, we used the publicly available pre-trained GloVe model containing word embeddings of one hundred dimensions each. For Juniper's datasets, we used the embedding matrix created and trained by Juniper Networks on their internal datasets. The Juniper Network's embedding matrix had vectors of one hundred and fifty dimensions each. The dictionaries of the word with index mappings, the embedding matrix, and the descriptive information about the dataset (such as the number of input words and the maximum length of the input string) were stored in a Python dictionary for later use in the model.

## **Embedding layer**

The embedding layer was a precursor layer to our model. This layer utilizes the embedding matrix, saved in the previous step of preparing the dataset, to transform each word to its vector form. The layer takes input each sentence represented in the form of word indexes and outputs a vector for each word in the sentence. The dimension of this vector is dependent on the embedding matrix used by the layer. Representing each word in a vector space is important because it gives each word a mathematical context and provides a way to calculate similarity among them. By representing the words in the vector space, our model can run mathematical functions on them and train itself.

## Model

Recent studies have shown that the sequence-to-sequence model with encoder-decoder outperforms simple feedforward neural networks for text summarization (Nallapati, Zhou, Santos, Gulçehre, & Xiang 2016). Sequence to sequence models have been commonly used in machine translation in the past. Our model utilizes the sequence-to-sequence design. Figure 9 shows the training portion of our model.

The model is provided with an input text “I have a dog. The dog is very tall.” and the expected summary is “I have a tall dog”. In the first step the input gets converted to the index form, using the word to index dictionary saved, and then the embedding layer translates it to a vector. The output from the embedding layer is fed to the encoder which consists of three LSTM layers. The green boxes in the figure form the encoder set of LSTMs. The LSTM layers in the encoder take the input from the embedding layer and maintain an internal state. The internal state gets updated as each word in the vector form is fed to the encoder. Each layer takes as input the state processed by the previous LSTM layer.

Once the entire input is processed, the internal state is then transferred to the decoder to process. The decoder consists of three LSTM layers as well as a SoftMax layer to compute the final output. The decoder (shown as the red boxes in the Figure 9), initializes itself using the states computed by the LSTM layers in the encoder. It works similar to the encoder by maintaining a hidden state and passing it on for decoding the next input word.

During the training phase, the decoder takes as input the actual or the reference summary as shown in the Figure 9. The decoder’s job is to use each input word to update its hidden state and then predict the next word in the summary after passing it through the SoftMax layer. The SoftMax layer computes the probability (whether the word is the output word) of each available

word in the stored output vocabulary list. The predicted value is then compared with expected value which is the next word in the summary. Depending on the loss calculated by the model, the weights of the LSTM cells are updated through the classic backpropagation algorithm.

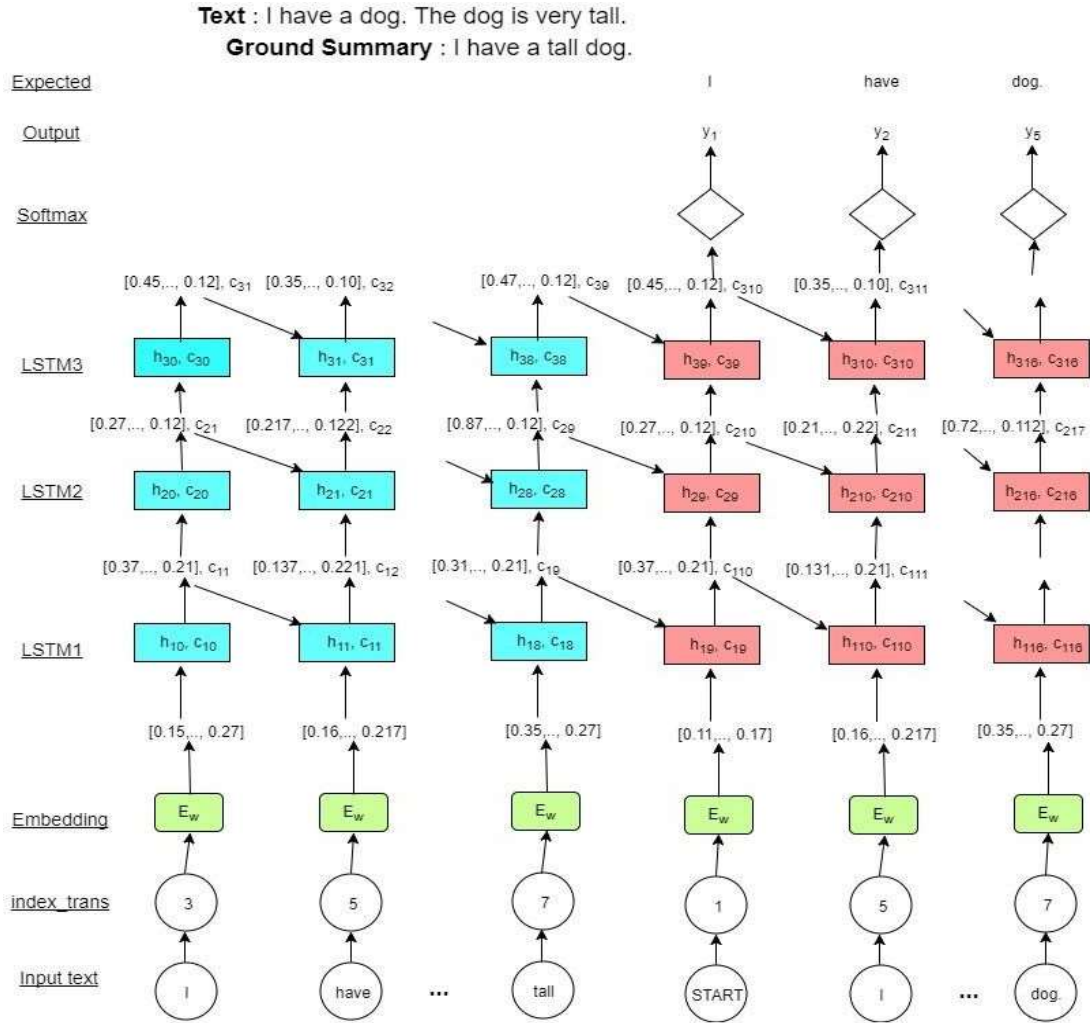


Figure 9: Architecture of training model

After the model has been trained properly on the three datasets, the model is evaluated on the portion of the dataset that was reserved for validation. For predicting the summary on the given input, the model is not shown the actual summary. The first input to the decoder is a 'start token' which is what the model was trained on. Once the model predicts a word, that word is

now used as the input to the decoder for the next word as shown in the Figure 10. Due to this, there is a slight difference between the training and the predicting stage of the model.

To overcome any discrepancies, during the training stage, we picked a random word as the input instead of the actual summary word one-tenth of the time as suggested in Generating news headlines with recurrent neural networks by Konstantin Lopyrev (2015). The softmax layer in the decoder outputs a one-hot encoding of each word. Using the mapping, the word is found and then joined together with rest of the output to produce a summary.

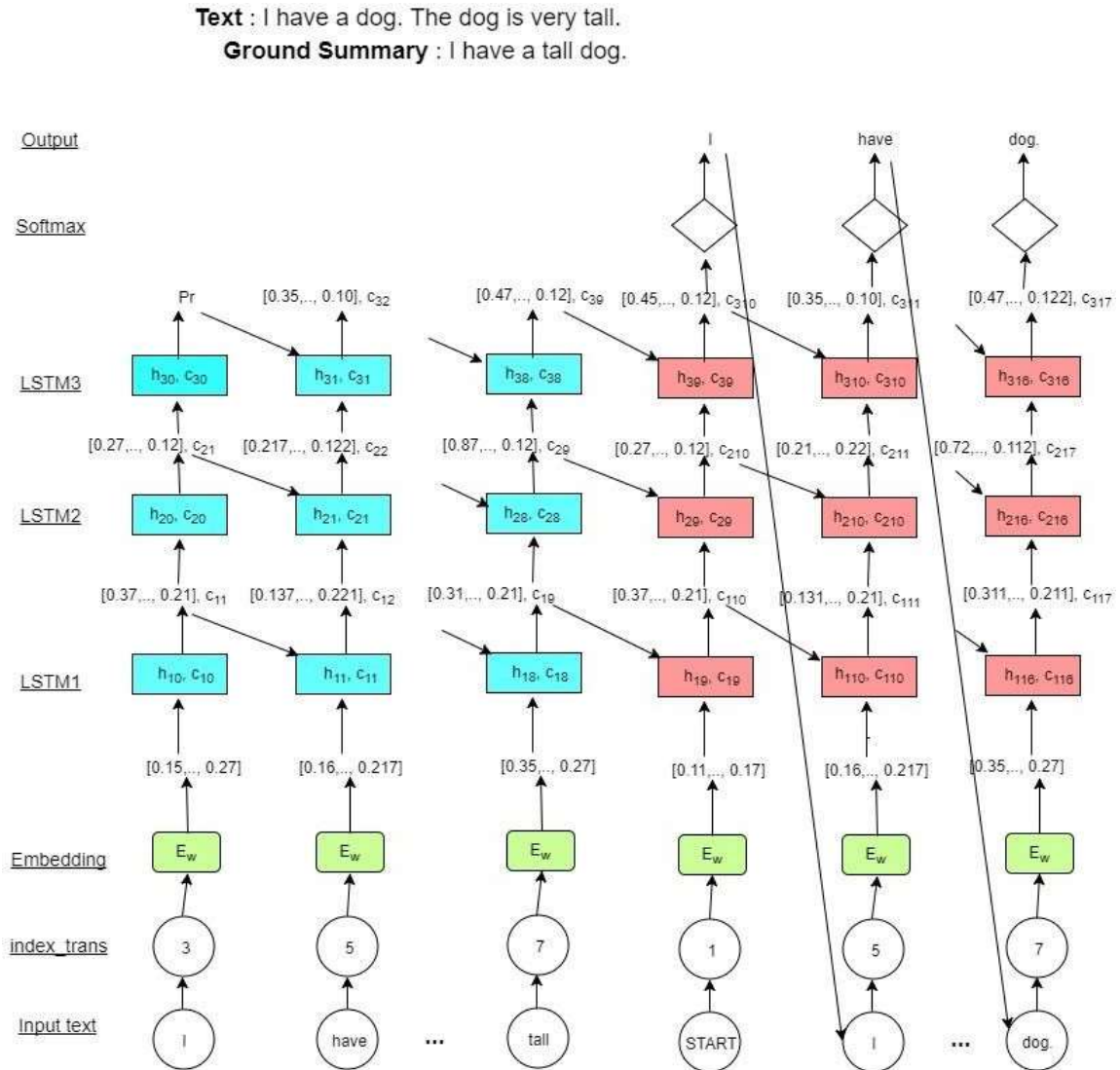


Figure 10: Architecture of prediction model

### Test and compare among different datasets

There were four different versions of the model created, each trained on one of the four cleaned datasets: the Stack Dataset, the KB Dataset, the JIRA Dataset and the JTAC Dataset. The abstractive summarization model was not trained on the News Dataset because the News Dataset is not closely related to Juniper's datasets, and training each model once can take up to 2 days. The four models were then tested on the portion of the datasets reserved for validation. The generated summaries were scored using ROGUE-1 and BLEU scores. Using these scores, the datasets capabilities of being summarized effectively can be measured. The results would show if the model works better on any particular dataset.

### Tune the model

After running and testing the model in different datasets, the model's parameters were tuned - specifically the number of hidden units was increased, the learning rate was increased, the number of epochs was changed and a dropout parameter (percentage of input words that will be dropped to avoid overfitting) was added at each LSTM layer in the encoder. Different values were tested for each of the parameters while keeping in mind the limited resources available to test the model. The models were rerun on the datasets, and the results were compared with the previous run. The summaries were also evaluated by human eyes and compared with the ones produced earlier. The best models were picked out which formed the backend of our system.

### Build an end-to-end application

Once the model was completed and tested, a web end-to-end application was built (for details, please see Section 5.5). The application was built in Python's flask library primarily because the models were implemented in Python. A bootstrap front-end UI was used to showcase the results. The UI consisted of a textbox for entering text, a dropdown for choosing the desired models for each of the three datasets and an output box for showing the results. The UI included a summarize button which would send the chosen options by making a POST AJAX request to the backend. The backend server would run the text on the pre-trained model and send the result back to the front-end. The front-end displayed the result sent by the backend. This application was the final product of this project which was hosted on a web server and can be viewed by any modern web browser.

## Results

This section displays the results we got from the data cleaning (Section 5.1), extractive models (Section 5.2), and abstractive models (Section 5.3 & Section 5.4). Then, Section 5.5 presents the end-to-end application built based on the abstractive model. With evaluation functions ROUGE-1 and BLEU, as described in methodology Section 4.4, we evaluated and compared our models' performance, and measured the level of success our model had in meeting our goals.

## Data Cleaning and Categorizing

After cleaning our datasets by removing code snippets, unrecognized words, non-English articles and extremely short articles, we got more human-readable and cleaner sentences as our model's input. Figure 11 shows one of the original solution text that contains code part after the `<code>` tag, many HTML tags such as `<br />`, unrecognized words such as “&nbsp” and unwanted links.

```
'To retrieve your IDP subscription license, please make sure you have applied your IDP subscription authorization code to your SRX serial number using the <a href="http://tools.juniper.net/subreg/">Subscription Registration Tool</a>.<br /><br />Ensure your device has internet connectivity, and access the CLI. From the CLI run the following command:<blockquote><div><code>request system license update</code><strong><br /></strong></div></blockquote><p>The unit will retrieve its license with the entitled dates from the server. You must reset the device after the key has been loaded<strong><br /></strong></p>If you need additional assistance, contact <a target="_blank" href="http://www.juniper.net/support/requesting-support.html">Customer Care</a> to open a case.<br /><ul><li>Web (online):&nbsp; Click <a target="_blank" href="https://tools.online.juniper.net/cm/case_create_choice.jsp">Create Case</a>, and choose the 'Customer Care Case' radio button.</li><li>Phone:&nbsp;&nbsp;  Call 1-888-314-JTAC (&nbsp;5822 )&nbsp;  (US and Canada) or &nbsp;1-408-936-1572 (outside the United States).&nbsp;&nbsp;&nbsp;&nbsp;  Select Option 2.</li></ul><br /><br />Spanish version, see <a target="_blank" href="http://kb.juniper.net/KB16767">KB16767</a>'
```

Figure 11: Example of Raw Solution Text



Figure 12 shows the solution text in Figure 11 after cleaning. All the code parts, unrecognized words, and tags are removed.

"To retrieve your IDP subscription license, please make sure you have applied your IDP subscription authorization code to your SRX serial number using the Subscription Registration Tool. Ensure your device has internet connectivity, and access the CLI. From the CLI run the following command: The unit will retrieve its license with the entitled dates from the server. You must reset the device after the key has been loaded. If you need additional assistance, contact Customer Care to open a case. Web (online): Click Create Case, and choose the 'Customer Care Case' radio button. Phone: Call 1-888-314-JTAC (5822) (US and Canada) or 1-408-936-1572 (outside the United States). Select Option 2. Spanish version, see KB16767"

Figure 12: Example of Cleaned Solution Text

For data categorizing, as it is shown in Figure 13, we got a list of longest common substrings among all KB Dataset categories. The circled categories are some examples of the parent level category names. For example, the parent category "CUSTOMER\_CARE" was listed, while the child category "CUSTOMER\_CARE\_1" was not. By manually looking into the list, we found that most of the category names were decent such as "HARDWARE", "FIREWALL" and "NETSCREEN". Notice that some Juniper product series' category names were also captured by the algorithm such as "SSG", "QFX" and "WXC". After extracting the categories, the top 30 parent categories were picked, and a hierarchy map was built.

```
[ 'AAA_802_1X', 'ACX', 'AD', 'AI', 'AL', 'ANTI', 'AR', 'ARP', 'AT', 'AX',
  'BGP', 'BR', 'BX', 'CACHING', 'CBF', 'CCC', 'CE', 'CO', 'CPE', 'CSPF',
  'CTP', 'CUSTOMER_CARE', 'CX', 'C_', 'DCM', 'DEEP INSPECTION', 'DHCP', 'DNS',
  'DSCP', 'DVMRP', 'DX', 'DX_', 'ER', 'ERING', 'ERX', 'EX', 'E_S',
  'FBF', 'FIREWALL', 'FRAME_RELAY', 'FTP', 'GLOBALPRO', 'GRE', 'H323',
  'HARDWARE', 'HDLC', 'I2J_TOOL', 'IC', 'IDP', 'ING', 'INT', 'IP', 'IS',
  'G_', 'IVE', 'JAVA', 'JCS', 'JSA', 'JUNOS', 'JW', 'J_SERIES', 'KBTV', 'L',
  '2TP', 'LA', 'LAYER', 'LD', 'LI', 'LN', 'M5', 'M7I', 'MA', 'MBGP', 'MCG_5',
  '000', 'MEDIA_FLOW', 'MIBS', 'MLPPP', 'MO', 'MOBILE', 'MOBILENEXT', 'MPL',
  'S', 'MSDP', 'MULTICAST', 'MVPN', 'MWS', 'MX', 'NAT PAT', 'NET', 'NETSCREE',
  'N', 'NS', 'NSM', 'OAC', 'ODYSSEY_SERIES', 'ON', 'OS', 'PL', 'PO', 'PP',
  'PPP', 'PRO', 'PTX', 'P_', 'QFX', 'QFX', 'QOS', 'RADIUS', 'RE', 'RI', 'R',
  'SVP', 'SA', 'SBR', 'SC', 'SD', 'SECURI', 'SERIES', 'SIP', 'SM', 'SNMP',
  'SR', 'SS', 'SSG', 'SSI', 'ST', 'SWITCH', 'S', 'T1600_1', 'T320', 'TA',
  'TCA', 'TE', 'TER', 'TFTP', 'TI', 'TIME SYNCHRONIZATION', 'TOOLS', 'TRA',
  'TUNNELING_PROTOCOLS', 'T_TOOL', 'VF', 'VGW', 'VIRTUAL_APPLIANCE', 'VR',
  'RP', 'VSTRM', 'VULNERABILITIES', 'VXA', 'WA', 'WI', 'WL', 'WLA', 'WSAM',
  'WX', 'WXC', 'XAUTH', 'XRE200', '_BSR', '_DE', '_S', '_SERIES']
```

Figure 13: The List of Longest Common Substrings among KB Categories

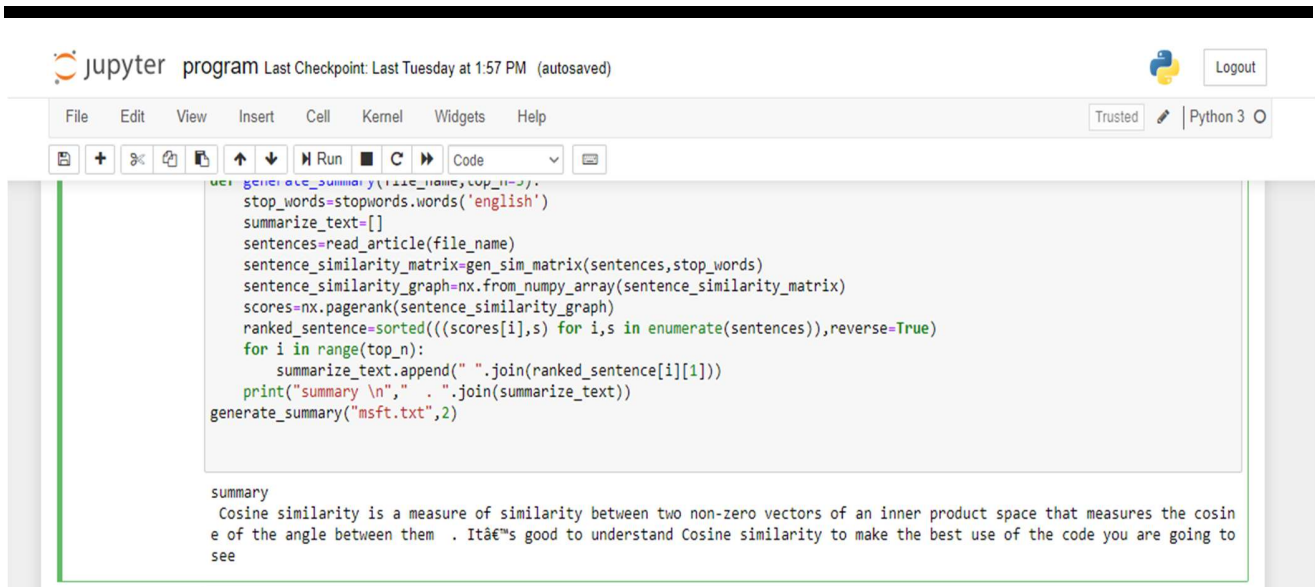


## TEXT SUMMARIZATION USING NLP

```
jupyter program Last Checkpoint: Last Tuesday at 1:57 PM (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [7]: import nltk
        #nltk.download('stopwords')
        from nltk.corpus import stopwords
        from nltk.cluster.util import cosine_distance
        import numpy as np
        import networkx as nx
        def read_article(file_name):
            file=open(file_name,"r")
            filedata=file.readlines()
            article=filedata[0].split(" ")
            sentences=[]
            for sentence in article:
                sentences.append(sentence.replace("[^a-zA-Z]", " ").split(" "))
            sentences.pop()
            return sentences
        def sentence_similarity(sent1,sent2,stopwords=None):
            if stopwords is None:
                stopwords=[]
            sent1=[w.lower() for w in sent1]
            sent2=[w.lower() for w in sent2]
            all_words=list(set(sent1+sent2))
            vector1=[0]*len(all_words)
            vector2=[0]*len(all_words)
            for w in sent1:
                if w in stopwords:
                    continue
```

```
jupyter program Last Checkpoint: Last Tuesday at 1:57 PM (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
        continue
        vector2[all_words.index(w)]+=1
        return 1-cosine_distance(vector1,vector2)
    def gen_sim_matrix(sentences,stop_words):
        similarity_matrix=np.zeros((len(sentences),len(sentences)))
        for idx1 in range(len(sentences)):
            for idx2 in range(len(sentences)):
                if idx1==idx2:
                    continue
                similarity_matrix[idx1][idx2]=sentence_similarity(sentences[idx1],sentences[idx2],stop_words)
        return similarity_matrix
    def generate_summary(file_name,top_n=5):
        stop_words=stopwords.words('english')
        summarize_text=[]
        sentences=read_article(file_name)
        sentence_similarity_matrix=gen_sim_matrix(sentences,stop_words)
        sentence_similarity_graph=nx.from_numpy_array(sentence_similarity_matrix)
        scores=nx.pagerank(sentence_similarity_graph)
        ranked_sentence=sorted(((scores[i],s) for i,s in enumerate(sentences)),reverse=True)
        for i in range(top_n):
            summarize_text.append(" ".join(ranked_sentence[i][1]))
        print("summary \n"," ".join(summarize_text))
    generate_summary("msft.txt",2)
```

## TEXT SUMMARIZATION USING NLP



The screenshot shows a Jupyter Notebook interface. At the top, the title bar says "jupyter program" and "Last Checkpoint: Last Tuesday at 1:57 PM (autosaved)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The toolbar shows icons for saving, adding cells, zooming, and running code. The code cell contains the following Python code:

```
def generate_summary(file_name, top_n=3):
    stop_words=stopwords.words('english')
    summarize_text=[]
    sentences=read_article(file_name)
    sentence_similarity_matrix=gen_sim_matrix(sentences,stop_words)
    sentence_similarity_graph=nx.from_numpy_array(sentence_similarity_matrix)
    scores=nx.pagerank(sentence_similarity_graph)
    ranked_sentence=sorted(((scores[i],s) for i,s in enumerate(sentences)),reverse=True)
    for i in range(top_n):
        summarize_text.append(" ".join(ranked_sentence[i][1]))
    print("summary \n"," ".join(summarize_text))
generate_summary("msft.txt",2)
```

The output cell shows the following text:

```
summary
Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them . It's good to understand Cosine similarity to make the best use of the code you are going to see
```

### Extractive Model Performance

The extractive summarization algorithms were tested on three different datasets (the News Dataset, the Stack Dataset, and the KB Dataset). Each dataset was scored on four different experiments - the first two experiments were run with the two chosen algorithms, Textrank and TF-IDF, while the last two experiments were the control experiments as described in Section 4.2.2.

Figure 14 shows the results of running the experiments in the News Dataset. As shown in the result, the news seemed to perform best when using the first three lines as the summary. In a news article, the first few lines, also called the leading paragraph, usually capture the content.

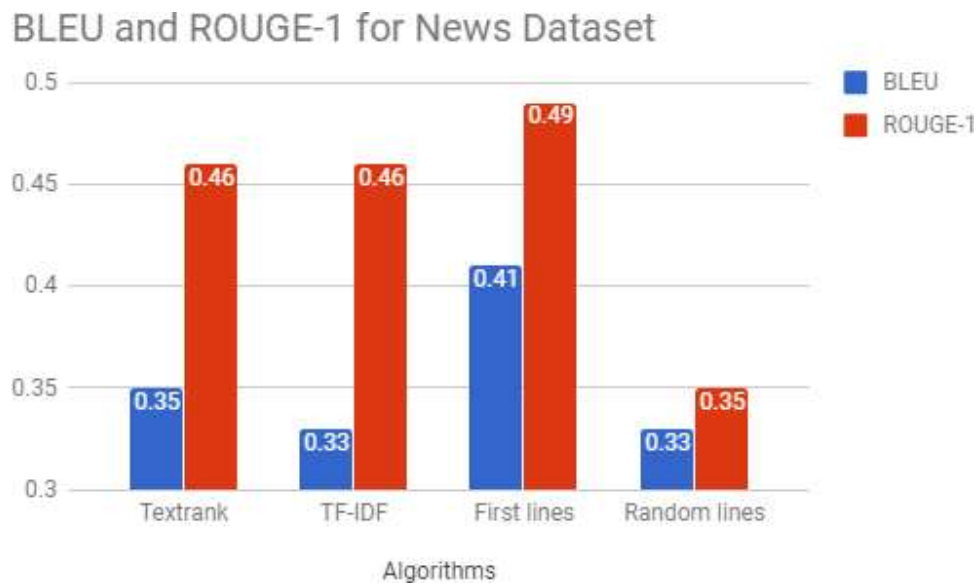


Figure 14: Extractive result from the News Dataset

Figure 15 shows the results of running the experiments on the Stack Dataset. As shown in the figure, the performance of Textrank and TF-IDF on the Stack Dataset was similar. They both performed better than the control experiments. However, summarizing the Stack Dataset by

extraction may often not make sense because some statements fall out of context. For example, if an article is describing steps to fix a broken web server, it might not help to extract a couple of sentences in the middle as a summary. An abstractive solution may be a better approach as instead of getting essential sentences from the steps; it should be summarized like “this article describes the steps to fix a broken web server”.

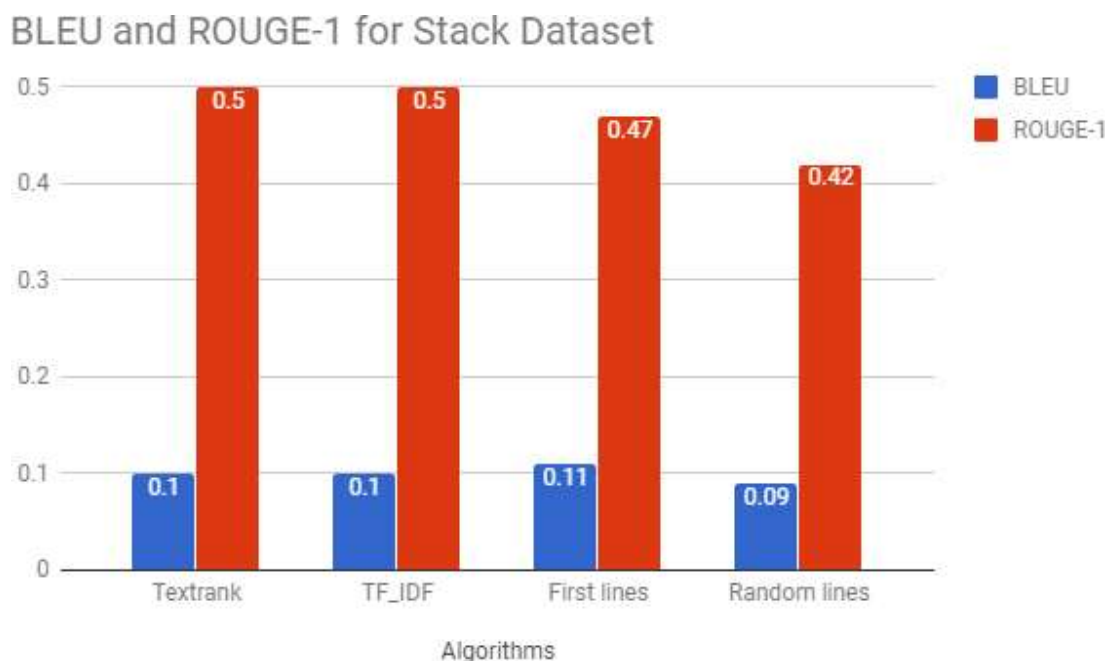


Figure 15: Extractive result from the Stack Dataset

The best algorithm for summarizing the KB Dataset by extraction was TF-IDF (as shown in Figure 16). We believe that this is because Juniper’s articles and their summaries often contain keywords which are unique to the article describing a specific Juniper product. TF-IDF takes into account how unique a word is to that document and then computes the importance of the sentence. However, similar to the results from the Stackoverflow dataset, we believe that summarizing Juniper’s articles, which often resembles a user guide or a manual, would be best

done by abstraction rather than by extraction. This is because the articles usually contain procedural steps and gathering the most important steps would often not capture the summary of the article. Later on, we decided to use TF-IDF to extract few sentences from each article in the KB Dataset and use that as input instead of the entire article for the abstractive summarization. The results of which will be discussed later.



Figure 16: Extractive result from the KB Dataset

## Abstractive Model Performance

The abstractive summarization algorithm was tested on four different datasets (the Stack Dataset, the KB Dataset, the JIRA Dataset, and the JTAC Dataset). The abstractive summarization model was configured to use 512 hidden units in each layer while training all four datasets. Also, the model was further configured such that any input data would be divided into batches of 64 articles each. Using batches, instead of the entire input dataset, helps increase the

computing speed. Before training the model, 5 percent of each dataset was saved to be a validation set in order to test our model's performance on different datasets. The rest 95 percent of the data was separated into the training set and the test set, and the two sets were used in the training process. Each model took about 2 days to be trained on the dataset. Once trained, the model can generate a summary within a fraction of a second. The rest of this section will present the results in detail for each dataset.

### **Stack Overflow Dataset: Question Body to Title**

For the Stack Overflow dataset, the model assumes that the titles are the summaries of the question bodies. The model was trained on 50 epochs, where each epoch means training the model through the entire dataset once. Running more epochs allow the model to improve its accuracy. Figure 17 shows the changes in categorical accuracy and categorical loss by each epoch. Categorical accuracy and categorical loss are evaluation methods provided by Keras (for details, see Appendix A: Expand Technical Terms).

For the model, the expected results for the training set are increasing accuracy and decreasing loss. However, after around 40 epochs, the accuracy and loss do not change much since the model has already become familiar with the input dataset. The graph tells us that the model is properly trained since the accuracy of the training set is increasing as expected.

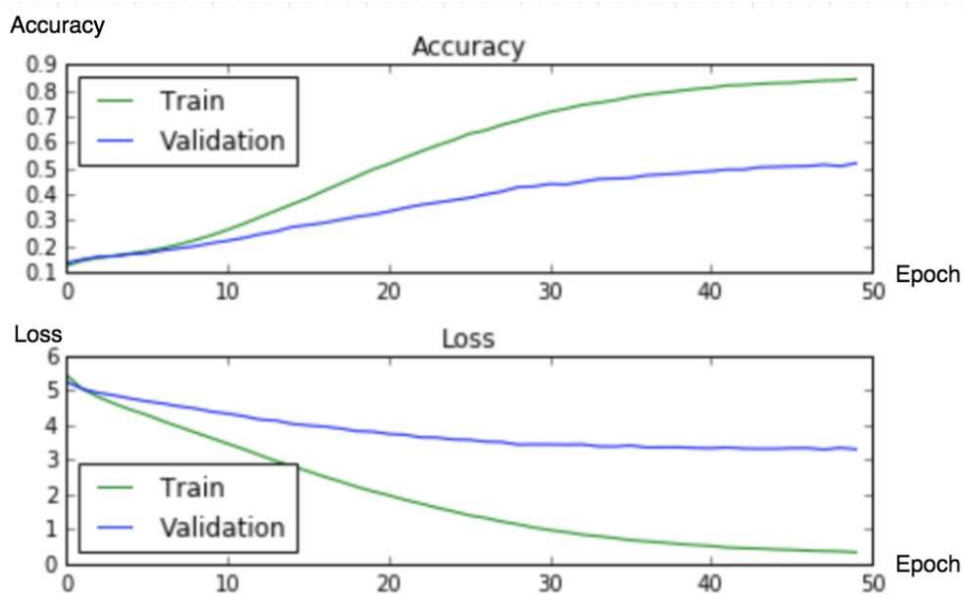


Figure 17: Stack Dataset: Categorical Accuracy vs. Epoch & Categorical Loss vs. Epoch

After training, we record the predictions produced from the validation set. Table 1 presents some manually picked good predictions. The 'END' is the symbol for the end of prediction. It is not a part of the generated summary.

Table 1: Good Predictions from the Stack Dataset - Question body to Title

Input Article (Question body)	Original Summary (Title)	Generated Summary (Title)
<p>I'm using jQuery for the first time and so I've added the jQuery 2.1 library to my ASP.NET project and referenced it from the Master Page like so:</p> <p>With all the files on the web server, the pages work fine in Chrome - all of the jQuery stuff functions as expected. However, if I load the same pages in IE11 (I haven't had a chance to test other browsers yet), none of the jQuery functionality works.</p> <p>Interestingly, if I launch IE11 from Visual Studio 2013's development environment, so that it runs the website in IIS express on my dev machine for testing purposes, everything works fine. So, it's clearly not a browser compatibility issue, and it seems that I'm doing things right since it works in Chrome on the web server.</p> <p>Any ideas what I could try? Are there any quirks for getting jQuery working in IE? I've tried referencing the jQuery script file from the individual Web Forms as well as the Master Page, but that didn't make any difference.</p>	jQuery Not Working in IE11 On Live Server	Jquery ajax callback with i.e. issue END
<p>I want to create a simple java web service. What tools I need to get started? What do I need to know to get started deploying this in Tomcat?</p>	How can I get started creating a Java web service?	how to create a webservice service in java END
<p>I've an error with jQuery on the first line of: I got another error with /jquery-1.5.1.js on line 3539:</p> <p>Everything I am working on FF, Chrome and Safari but I got errors on IE. The errors are: "Object doesn't support this property or method"</p>	Error with JQuery on line 3539 with IE	Jquery ui modal doesn't work in i.e. END



## Juniper Knowledge Base Dataset: Solution to Title

For the Juniper KB Dataset, the model assumes that the titles are the summaries of the solutions. In some cases, the title may represent the question that is being answered by the article instead of the actual summary of the solution. However, our model requires summaries to be trained on and because the titles captured the essence of the solution, they were considered as summaries for our purposes. Figure 18 shows the accuracy and loss changes over epochs.

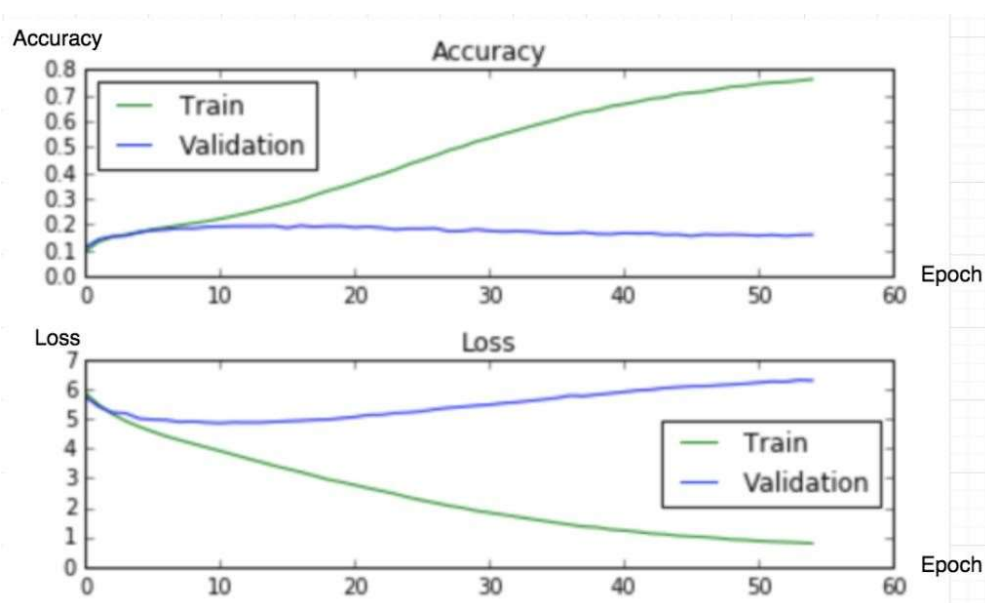


Figure 18: KB Dataset - Solution to Title: Categorical Accuracy vs. Epoch & Categorical Loss vs. Epoch

In Figure 18, the validation set did not show a consistent increase in accuracy and a consistent decrease in the loss. This is because although our model generates some good results, it is not perfect yet. So, unlike the training set, the validation set was not as consistent with its improvements (some suggestions for making the model perform better is included in Section 6.0). Table 2 shows some manually picked out good predictions. Due to the nature of the KB Dataset, it might be difficult for non-Juniper employees to understand some of the input articles.

Table 2: Good Predictions from the KB Dataset - Solution to Title

Input Article (Solution)	Original Summary (Title)	Generated Summary (Title)
M160 FRUs: Hot swappable: Routing Engine Switching and Forwarding Module (SFMs) Flexible PIC Concentrator (FPCs) Type 1 and Type 2 Physical Interface Card (PICs) Miscellaneous Control Subsystem (MCSs) PFE Clock Generator (PCGs) Power Supplies Air Filter Rear Bottom Impeller Rear Top Impeller Front Impeller (Craft Interface is part of this FRU) Fan Tray (Cable Management System is part of this FRU) Requires Power Down: Circuit Breaker Box Connector Interface Panel (CIP) M160 Chassis (includes backplane) Please refer to the M160 hardware guide for more detail.	[Archive] What are the specific Field Replaceable Units (FRU) on the M160?	archive what are the specific field replaceable units fur on the m e
RSVP Interface Highwater Mark 'Highwater mark' indicates the highest bandwidth that has ever been reserved on this interface, in bps.	[Archive] What is an RSVP interface Highwater Mark?	archive how to configure rsvp over an forward routes
Q: Why does the debug show packet dropped, denied by policy? A: In a route-based VPN, if the tunnel interface is created in a different zone than that of the user traffic, a policy is required. Recall, when crossing zones, a policy needs to be configured. See KB6551 for additional details on tunnel zones and policies	What does the Debug Message "packet dropped, denied by policy" mean?	Screen os understanding the policy id in debug mode

### Juniper Knowledge Base: Extracted Solution to Title

We trained another model on the KB Dataset using the extracted solution as the input for the model instead of the entire solution of each article. The extracted solution was gathered by applying our text extraction algorithms on the KB Dataset. As explained in Section 3.0, the KB Dataset is not as clean as public datasets like the news articles dataset. Since the extractive summarization tool, described in Section 4.2, is capable of capturing the key points in the

solution and generating a neater text input, the model was fed the extracted solution to predict the title of the article. Figure 19 shows the accuracy and loss changes over epochs for the training of this model.

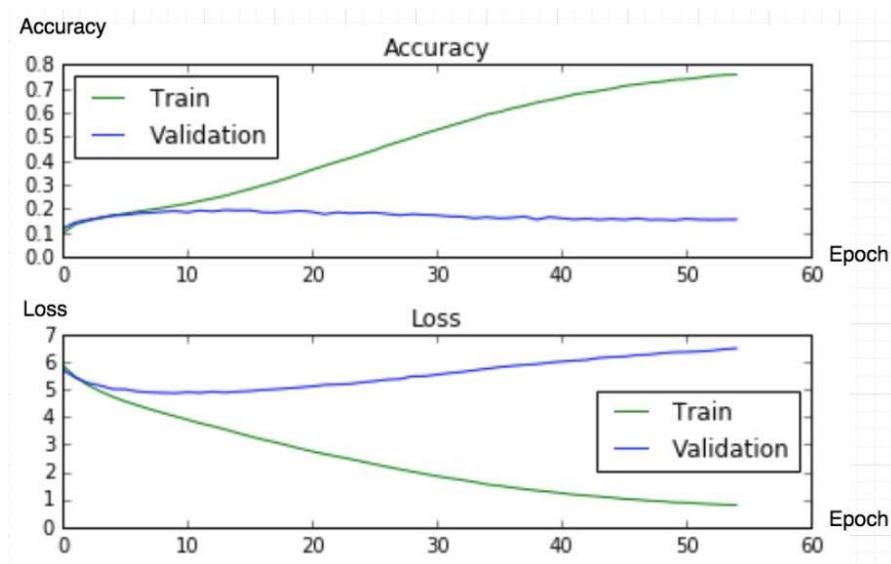


Figure 19: KB Dataset - Solution (Extracted) to Title: Categorical Accuracy vs. Epoch & Categorical Loss vs. Epoch

Table 3 shows some manually picked out good predictions. From the table, we can see that the generated summaries in some examples, like the first row in Table 3, capture important keywords presented in the original summary, but fails to correctly summarize the article properly. One explanation for this is the limited data we were used. Usually, for tasks like text summarization, millions of cases are required in order to generate decent results (Lopyrev, 2015).

Table 3: Good Predictions from the KB Dataset - Solution (Extracted) to Title

Input Article (Solution)	Original Summary (Title)	Generated Summary (Title)
So, the HC Policy has to be defined as Check the process is running Check the file with MD5 checksum Hence IVE administrator would have to define both the Host Checker policies and implement at the role or realm level. One work around is to combine Process Checking and File MD5 Checksum. This issue applies until IVE version 6.	Host Checker fails with " Can't Generate Checksum" error [Applicable until IVE version 6.1rx]	host checker fails error connect due to user side and user login
M160 FRUs Hot swappable Routing Engine Switching and Forward... Requires Power Down Circuit Breaker Box Connector Interface Panel CIP M160 Chassis includes backplane Please refer to the M160 hardware guide for more detail.	[Archive] What are the specific Field Replacable Units (FRU) on the M160?	archive what are the different field replacable units fru on the m e

### Juniper JIRA Dataset: Description to Summary

To see the performance of the model on datasets from Juniper Networks other than the KB Dataset, the model was also trained on the JIRA Dataset. For the JIRA Dataset, the input for the model was the description of the article and the expected output was the given summary in the article. Figure 20 shows the accuracy and loss change over epochs for the training of this model. The graph in Figure 20 shows results with trends similar to the results of the KB Dataset. Therefore, it means the model had a similar performance when training on both datasets.

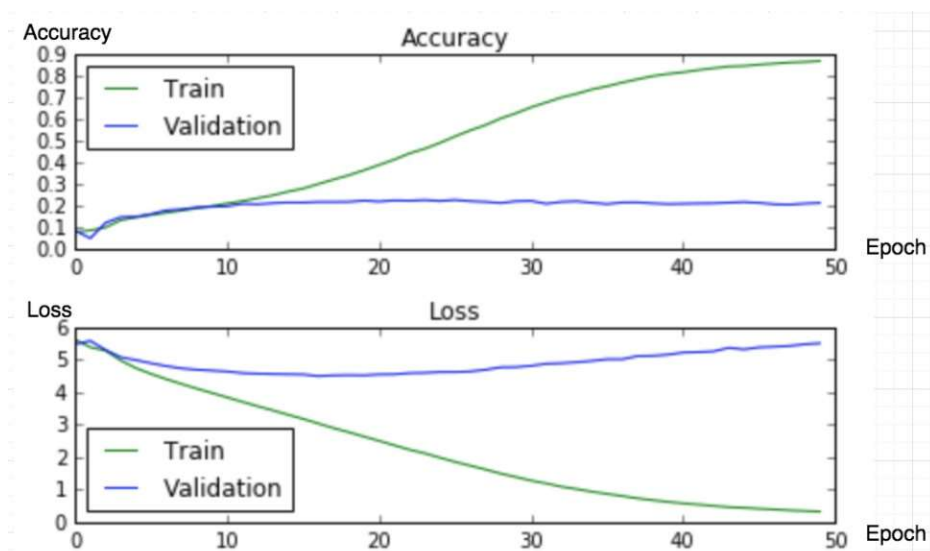


Figure 20: JIRA Dataset: Categorical Accuracy vs. Epoch & Categorical Loss vs. Epoch

Table 4 are some manually picked out good predictions. Similar to what we explained before, although the summary generated by the model contains some of the important keywords, it does not align very well with the actual summary of the article.

Table 4: Good Predictions from the JIRA Dataset

Input Article (Description)	Original Summary (Summary)	Generated Summary (Summary)
File names are appearing in attachment details page twice also blank file names observed. PFA screenshot	File names are appearing in attachment details page twice also blank file names observed	file name is not displaying
In RMA Task details page, the notes API is not being called on clicking "Refresh" button.	rma details need to refresh the rma number of table view	[RMA Details] In Task details page, the notes API is not being called on clicking "Refresh" button

## TEXT SUMMARIZATION USING NLP

---

Tried testing the API (POST) /api/download- manager/download Attachments URI: <a href="http://.../api/download-&lt;br/&gt;manager/download">http://.../api/download- manager/download</a> Attachments	/api/download- manager/downloaded hments API returning "Failed to decode: Unrecognizedfield	error in querying case api
--	--	-------------------------------

## Juniper JTAC Dataset: Description to Synopsis

JTAC is another dataset from Juniper that has a human-generated synopsis. The model was trained with synopsis as the output and the description as input. Figure 21 shows the accuracy and loss change over epochs.

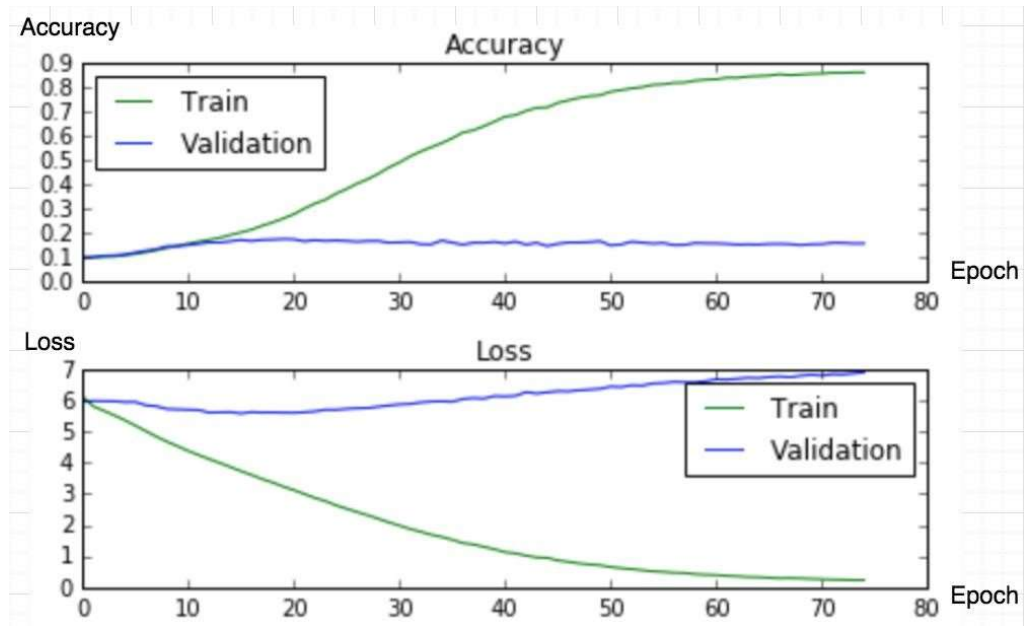


Figure 21: JTAC Dataset: Categorical Accuracy vs. Epoch & Categorical Loss vs. Epoch

Table 5 shows some manually picked out good predictions. Some descriptions have been hidden because it is too long for the table. The model shows some good predictions like the generated summary in the first row matches very well with the original summary. In some cases, even though the summary generated by the model has words different from the actual summary, the correct idea is still preserved by the generated summary.

Table 5: Good Predictions from the JTAC Dataset

Input Article (Description)	Original Summary (Synopsis)	Generated Summary (Synopsis)
hi support please see alarm root mgojunre show chassis alarms no forwarding alarms currently activealarm time class description pht major fpc major errors error code attached rsi ... remained clean closing techn ical case per support needed customer contact agreed close marione jgramirez	FPC 5 Major Error - Error code 65537	fpc major errors
high cpu utilization observed one routers cpu utilization user percent eopqqtu hrstwi show system processes extensive ... hyperlink idkb customer contact pushkar sanchit tarun	rma details need to refresh the rma number of table view	[RMA Details] In Task details page, the notes API is not being called on clicking "Refresh" button
switch chassis member issue integrating switches issue description ex continuously creates ... current network status resolved resolution detail pr customer contact elad	they have 3 switch 2 chassis is not member have issue integrating the switches	switch not functioning
hello one power supplies failed mx router beginning saw bunch warnings like pemaltiusgettemp ... archive case issue description issue description pem faulty current network status stable resolution detail processed rma rma delivered customer contact sergey yemelyanovich	Power Supply failed	ex power supply down
routing engines visible forwarding table issue description routes seen forwarding table routing instance current network statusissue resolved configuration changes resolution detail instance type specified routing instance caused forwarding tableshow routes customer contact jacek	routing engines are not visible in the forwarding table	routing engine issue



## Performance Comparison Among Datasets

Figure 22 shows the comparison among the model's performances on different datasets in terms of ROUGE-1 and BLEU scores. Figure 22 shows that our model has the best performance on the JIRA Dataset. The two metrics ROUGE-1 and BLEU scores are computed by comparing the generated summary with the actual summary word by word. However, abstractive summarization is not about creating the same summary as the actual summary, because two summaries can be different when compared word by word while still summarizing the article properly. So, while the two metrics give some context about the generated summary, it is important to manually look at some of the generated summaries and not base our evaluation of the model completely on the scores of the two metrics.

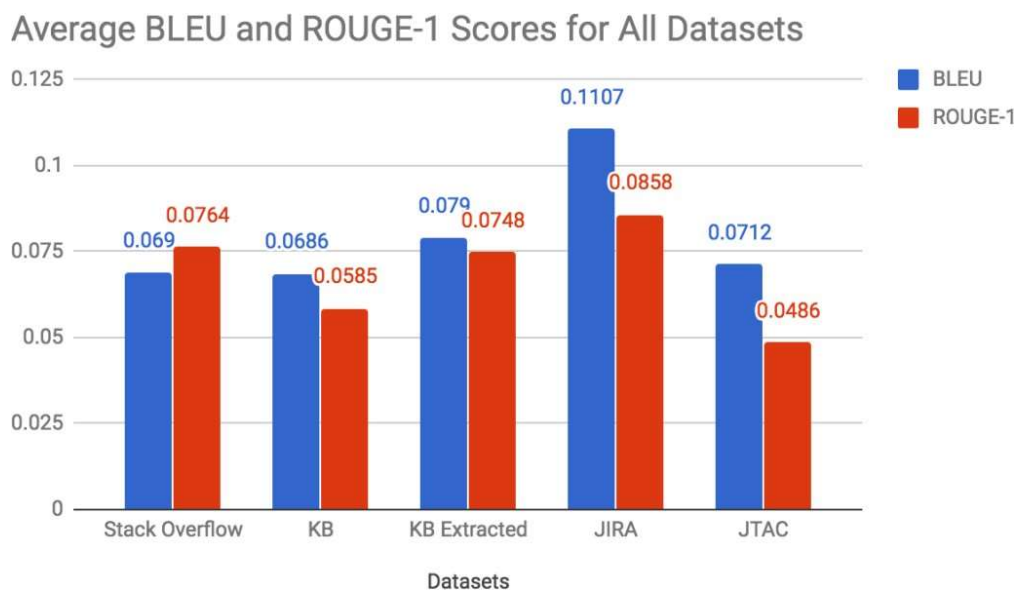


Figure 22: Abstractive Summarization performance over different datasets

Due to our limited resources, the small size and noisy datasets, our scores do not reflect great performance. On an average, with our resources, a model training on 15,000 articles over

50 epochs with a vocabulary of 50,000 words and 512 hidden units took about 2 days to complete training. Ideally, we should be training on at least a million articles with a much larger vocabulary to get a decent performance. That being said, as shown above, we were able to generate a few good results and with our findings, we believe while it may be difficult to implement text summarization in Juniper Networks' datasets directly, text summarization shows promising results and can be a huge advancement for Juniper Networks and the tech industry in general.

### **End-to-End Application**

Finally, we developed an end-to-end web application to demonstrate the concept of text summarization (Figure 23). The web application was hosted on a Microsoft Azure web server. Once logged into the web app, one could enter an input text, select the appropriate model they want the text to run on and get back the summary. The once the user clicks the summarize button, the HTML page makes a POST request to the backend server which has preloaded all the appropriate models. After the request is received, the server runs the input against the model, fetches the result and sends it as the response to the web client. The web page is then updated to reflect the output. Developing an end-to-end application helped demonstrate the results and the capabilities of text summarization efficiently. The backend server of the tool could be used by the chatbot to fetch real-time summaries once the model is accurate enough.

## Abstractive Summary

### Input article:

I want to create a simple java web service.  
What tools I need to get started?  
What do I need to know to get started deploying this in Tomcat?

Model: StackOverflo ▾

Summarize

### Summary Output:

how to create a webservice service in java

Figure 23: Screenshot of end-to-end text summarization web application

## Conclusion

With the ever-growing text data, text summarization seems to have the potential for reducing the reading time by showing summaries of the text documents that capture the key points in the original documents. Juniper Networks also has many large datasets that are still growing in size. Applying text summarization on each article can potentially improve customer experience and employees' productivity.

We worked on building text summarization tool on the public datasets and the datasets provided by Juniper Networks. By taking inspiration from previous works, we built two tools for text summarization. The first text summarization tool performs extractive summarization on the input articles using TF-IDF and Textrank. The extractive summarization tool allows extraction of any number of key sentences from the original articles. Another tool we implemented is abstractive summarization tool with neural networks. We built an encoder-decoder model using six LSTM layers: three layers in the encoder, and the other three in the decoder. The compiled models were trained on four similar datasets, and an end-to-end web application was built using the trained models. The end-to-end summarizer can perform text summarization for any input sentences, however, for best results, an input closely related to chosen dataset model is required.

In a nutshell, the tools we made explored the possibility of implementing text summarization on Juniper's dataset. While some datasets had decent summaries generated by our model, there are several ways the model can be improved further.

First, we think the datasets can be further cleaned. In our data cleaning process, we removed the articles containing Spanish words, however, there may be other articles containing non-English words. Any such words that cannot be recognized by the model should be removed in order to improve the model's performance. The code snippets denoted with `<code>` tags were

removed from the articles, but there may be other chunks of code in the text portion of the articles outside the `<code>` tags. Having such chunks in the input dataset will affect the model's performance in a negative way. We recommend looking into ways of detecting code written in various programming languages in an input text.

The model we built for abstractive summarization did a good job on generating human-readable sentences from given inputs. However, it did not always generate summaries capturing all the important information in the input documents. To solve this problem, based on our research, we propose adding a custom layer to the model that performs attention mechanism (Lopyrev, 2015). The attention mechanism has been proved to be useful in tasks like abstractive summarization.

Lastly, we suggest using larger datasets to train the models. Researchers in the past have trained their text summarization models on millions of documents to achieve good results (Nallapati, Zhou, Santos, Gulçehre, & Xiang 2016). Whereas, due to limited resources, the largest dataset we used only had about twenty thousand articles. If these changes can be applied, we think that the performance of the model may improve.

## Reference

- Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint arXiv:1409.0473*. Retrieved February 28, 2018.
- Brill, E. (2000). Part-of-speech Tagging. *Handbook of Natural Language Processing*, 403-414. Retrieved March 01, 2018.
- Brownlee, J. (2017a, November 29). A Gentle Introduction to Text Summarization. Retrieved March 02, 2018, from <https://machinelearningmastery.com/gentle-introduction-text-summarization/>
- Brownlee, J. (2017b, August 09). How to Use Metrics for Deep Learning with Keras in Python. Retrieved February 28, 2018, from <https://machinelearningmastery.com/custom-metrics-deep-learning-keras-python/>
- Brownlee, J. (2017c, October 11). What Are Word Embeddings for Text? Retrieved February 28, 2018, from [machinelearningmastery.com/what-are-word-embeddings/](https://machinelearningmastery.com/what-are-word-embeddings/)
- Chowdhury, G. (2003). Natural Language Processing. *Annual Review of Information Science and Technology*, 37(1), 51-89. doi:10.1002/aris.1440370103. Retrieved March 02, 2018
- Christopher, C. (2015, August 27). Understanding LSTM Networks. Retrieved March 02, 2018, from [colah.github.io/posts/2015-08-Understanding-LSTMs/](https://colah.github.io/posts/2015-08-Understanding-LSTMs/)
- Dalal, V., & Malik, L. G. (2013, December). A Survey of Extractive and Abstractive Text Summarization Techniques. In *Emerging Trends in Engineering and Technology (ICETET)*, 2013 6th International Conference on (pp. 109-110). IEEE. Retrieved March 01, 2018.
- Das., K. (2017). Introduction to Flask. Retrieved February 27, 2018, from [pymbook.readthedocs.io/en/latest/flask.html](http://pymbook.readthedocs.io/en/latest/flask.html)

- Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (pp. 249-256). Retrieved March 2, 2018, from <http://proceedings.mlr.press/v9/glorot10a.html>
- Hasan, K. S., & Ng, V. (2010, August). Conundrums in Unsupervised Keyphrase Extraction: Making Sense of the State-of-the-art. In Proceedings of the 23rd International Conference on Computational Linguistics: Posters (pp. 365-373). Association for Computational Linguistics. Retrieved February 28, 2018.
- Getbootstrap.com. (2018). Retrieved March 02, 2018, from <http://getbootstrap.com/docs/4.0/getting-started/introduction/>
- Juniper Networks. (2018). Retrieved March 02, 2018, from <https://www.juniper.net/us/en/>
- Keras: The Python Deep Learning library. (n.d.). Retrieved February 27, 2018, from <https://keras.io/>
- Ketkar, N. (2017). Introduction to Keras. In Deep Learning with Python (pp. 97-111). Apress, Berkeley, CA. Retrieved February 26, 2018, from [https://link.springer.com/chapter/10.1007/978-1-4842-2766-4\\_7](https://link.springer.com/chapter/10.1007/978-1-4842-2766-4_7).
- Lin, C. Y. (2004). Rouge: A Package for Automatic Evaluation of Summaries. Text Summarization Branches Out. Retrieved February 25, 2018.
- Lopyrev, K. (2015). Generating News Headlines with Recurrent Neural Networks. arXiv preprint arXiv:1512.01712. Retrieved February 28, 2018.
- LXML - Processing XML and HTML with Python. (2017, November 4). Retrieved February 25, 2018, from [lxml.de/index.html](http://lxml.de/index.html)

- Mihalcea, R., & Tarau, P. (2004). TextRank: Bringing Order into Text. In Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing. Retrieved February 27, 2018.
- Mohit, B. (2014). Named Entity Recognition. In Natural Language Processing of Semitic Languages (pp. 221-245). Springer, Berlin, Heidelberg. Retrieved February 27, 2018.
- Nallapati, R., Zhou, B., Gulcehre, C., & Xiang, B. (2016). Abstractive Text Summarization Using Sequence-to-sequence RNNs and beyond. arXiv preprint arXiv:1602.06023. Retrieved February 23, 2018.
- Natural Language Toolkit. (2017, September 24). Retrieved February 23, from <http://www.nltk.org/>
- Papineni, K., Roukos, S., Ward, T., & Zhu, W. J. (2002, July). BLEU: a Method for Automatic Evaluation of Machine Translation. In Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (pp. 311-318). Association for Computational Linguistics. Retrieved March 01, 2018.
- Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 1532-1543). Retrieved March 01, 2018.
- Python Data Analysis Library. (n.d.). Retrieved March 02, 2018, from <https://pandas.pydata.org/>
- Radhakrishnan, P. (2017, October 16). Attention Mechanism in Neural Network – Hacker Noon. Retrieved March 02, 2018, from <https://hackernoon.com/attention-mechanism-in-neural-network-30aaf5e39512>
- Rahm, E., & Do, H. H. (2000). Data Cleaning: Problems and Current Approaches. IEEE Data Eng. Bull., 23(4), 3-13. Retrieved March 01, 2018.



