WikiKreator: Improving Wikipedia Stubs Automatically

Siddhartha Banerjee
The Pennsylvania State University
Information Sciences and Technology
University Park, PA, USA
sub253@ist.psu.edu

Prasenjit Mitra
Qatar Computing Research Institute
Hamad Bin Khalifa University
Doha, Qatar
pmitra@qf.org.qa

Abstract

Stubs on Wikipedia often lack comprehensive information. The huge cost of editing Wikipedia and the presence of only a limited number of active contributors curb the consistent growth of Wikipedia. In this work, we present WikiKreator, a system that is capable of generating content automatically to improve existing stubs on Wikipedia. The system has two components. First, a text classifier built using topic distribution vectors is used to assign content from the web to various sections on a Wikipedia article. Second, we propose a novel abstractive summarization technique based on an optimization framework that generates section-specific summaries for Wikipedia stubs. Experiments show that WikiKreator is capable of generating well-formed informative content. Further, automatically generated content from our system have been appended to Wikipedia stubs and the content has been retained successfully proving the effectiveness of our approach.

1 Introduction

Wikipedia provides comprehensive information on various topics. However, a significant percentage of the articles are stubs[1] that require extensive effort in terms of adding and editing content to transform them into complete articles. Ideally, we would like to create an automatic Wikipedia content generator, which can generate a comprehensive overview on any topic using available information from the web and append the generated content to the stubs. Addition of automatically generated content can provide a useful starting point for contributors on Wikipedia, which can be improved upon later.

Several approaches to automatically generate Wikipedia articles have been explored (Sauper and Barzilay, 2009; Banerjee et al., 2014; Yao et al., 2011). To the best of our knowledge, all the above mentioned methods identify information sources from the web using keywords and directly use the most relevant excerpts in the final article. Information from the web cannot be directly copied into Wikipedia due to copyright violation issues (Banerjee et al., 2014). Further, keyword search does not always satisfy information requirements (Baeza-Yates et al., 1999). To address the above-mentioned issues, we present WikiKreator – a system that can automatically generate content for Wikipedia stubs. First, WikiKreator does not operate using keyword search. Instead, we use a classifier trained using topic distribution features to identify relevant content for the stub. Topic-distribution features are more effective than keyword search as they can identify relevant content based on word distributions (Song et al., 2010). Second, we propose a novel abstractive summarization (Dalal and Malik, 2013) technique to summarize content from multiple snippets of relevant information.[2]

Figure 1 shows a stub that we attempt to improve using WikiKreator. Generally, in stubs, only the introductory content is available; other sections ($s_1, ..., s_r$) are absent. The stub also belongs to several categories (C_1,C_2, etc. in Figure) on Wikipedia. In this work, we address the following research question: *Given the introductory content, the title of the stub and information on the categories - how can we transform the stub into a com-*

[1] https://en.wikipedia.org/wiki/Wikipedia:Stub

[2] An example of our system's output can be found here – https://en.wikipedia.org/wiki/2014_Enterovirus_D68_outbreak – content was added on 5th Jan, 2015. The sections on *Epidemiology*, *Causes* and *Prevention* have been added using content automatically generated by our method.

Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pages 867–877,
Beijing, China, July 26-31, 2015.

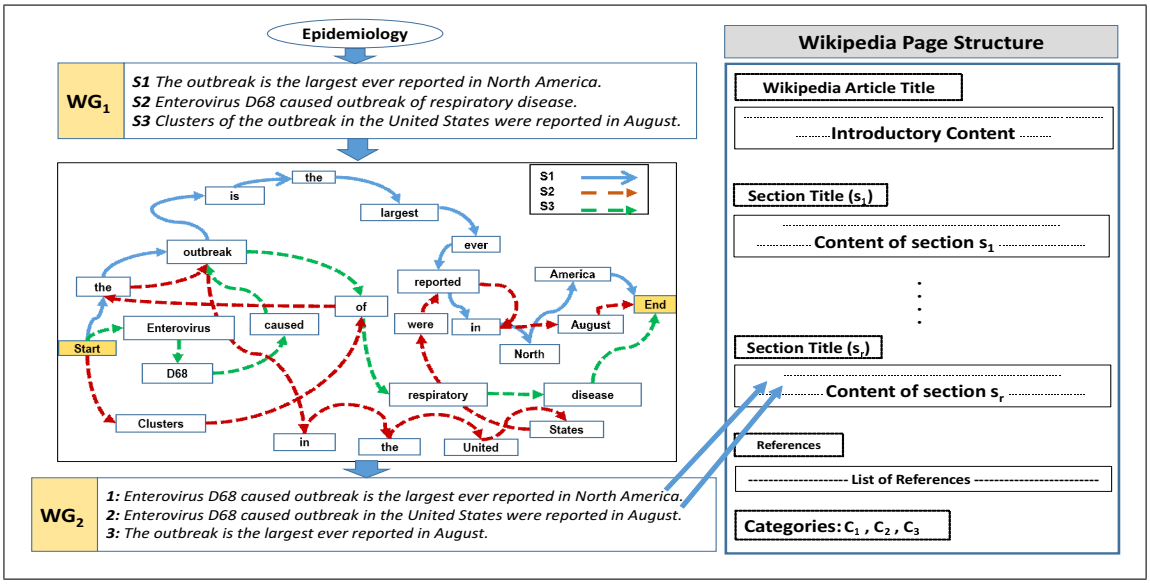

Figure 1: Overview of our word-graph based generation (left) to populate Wikipedia template (right)

prehensive Wikipedia article?

Our proposed approach consists of two stages. First, a text classifier assigns content retrieved from the web into specific sections of the Wikipedia article. We train the classifier using a set of articles within the same category. Currently, we limit the system to learn and assign content into the 10 most frequent sections in any given category. The training set includes content from the most frequent sections as instances and their corresponding section titles as the class labels. We extract topic distribution vectors using Latent Dirichlet Allocation (LDA) (Blei et al., 2003) and use the features to train a Random Forest (RF) Classifier (Liaw and Wiener, 2002). To gather web content relevant to the stub, we formulate queries and retrieve top 20 search results (pages) from Google. We use boilerplate detection (Kohlschütter et al., 2010) to retain the important excerpts (text elements) from the pages. The RF classifier classifies the excerpts into one of the most frequent classes (section titles). Second, we develop a novel Integer Linear Programming (ILP) based abstractive summarization technique to generate text from the classified content. Previous work only included the most informative excerpt in the article (Sauper and Barzilay, 2009); in contrast, our abstractive summarization approach minimizes loss of information that should ideally be in an Wikipedia article by fusing content from several sentences. As shown in Figure 1, we construct a word-graph (Filippova, 2010) using all the sentences (WG_1) assigned to a specific class (*Epidemiology*) by the classifier. Multiple paths (sentences) between the *start* and *end* nodes in the graph are generated (WG_2). We represent the generated paths as variables in the ILP problem. The coefficients of each variable in the objective function of the ILP problem is obtained by combining the information score and the linguistic quality score of the path. We introduce several constraints into our ILP model. We limit the summary for each section to a maximum of 5 sentences. Further, we avoid redundant sentences in the summary that carry similar information. The solution to the optimization problem decides the paths that are selected in the final section summary. For example, in Figure 1, the final paths determined by the ILP solution, – 1 and 2 in WG_2, are assigned to a section (s_r), where (s_r) is the section title *Epidemiology*.

To the best of our knowledge, this work is the first to address the issue of generating content automatically to transform Wikipedia stubs into comprehensive articles. Further, we address the issue of abstractive text summarization for Wikipedia content generation. We evaluate our approach by generating articles in three different categories: *Diseases and Disorders*[3], *American Mathematicians*[4] and *Software companies of the United States*[5]. Our LDA-based classi-

[3] https://en.wikipedia.org/wiki/Category:Diseases_and_disorders

[4] https://en.wikipedia.org/wiki/Category:American_mathematicians

[5] https://en.wikipedia.org/wiki/Category:Software_companies_of_the_United_States

fier outperforms a TFIDF-based classifier in all the categories. We use ROUGE (Lin, 2004) to compare content generated by WikiKreator and the corresponding Wikipedia articles. The results of our evaluation confirm the benefits of using abstractive summarization for content generation over approaches that do not use summarization. WikiKreator outperforms other comparable approaches significantly in terms of content selection. On ROUGE-1 scores, WikiKreator outperforms the perceptron-based baseline (Sauper and Barzilay, 2009) by ~20%. We also analyze reviewer reactions, by appending content into several stubs on Wikipedia, most of which (~77%) have been retained by reviewers.

2 Related Work

Wikipedia has been used to compute semantic relatedness (Gabrilovich and Markovitch, 2007), index topics (Medelyan et al., 2008), etc. However, the problem of enhancing the content of a Wikipedia article has not been addressed adequately. Learning structures of templates from the Wikipedia articles have been attempted in the past (Sauper and Barzilay, 2009; Yao et al., 2011). Both these efforts use queries to extract excerpts from the web and the excerpts ranked as the most relevant are added into the article. However, as already pointed out, current standards of Wikipedia requires rewriting of web content to avoid copyright violation issues.

To address the issue of copyright violation, multi-document abstractive summarization is required. Various abstractive approaches have been proposed till date (Nenkova et al., 2011). However, these methods suffer from severe deficiencies. Template-based summarization methods work well, but, it assumes prior domain knowledge (Li et al., 2013). Writing style across articles vary widely; hence learning templates automatically is difficult. In addition, such techniques require handcrafted rules for sentence realization (Gerani et al., 2014). Alternatively, we can use text-to-text generation (T2T) (Ganitkevitch et al., 2011) techniques. WikiKreator constructs a word-graph structure similar to (Filippova, 2010) using all the sentences that are assigned to a particular section by a text classifier. Multiple paths (sentences) from the graph are generated. WikiKreator selects few sentences from this set of paths using an optimization problem formulation that jointly maximizes the informativeness and readability of section-specific snippets and generates output that is informative, well-formed and readable.

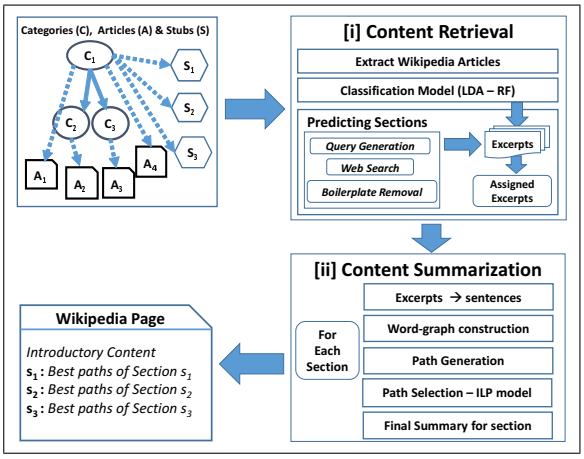

Figure 2: WikiKreator System Architecture: Content Retrieval and Content Summarization

3 Proposed Approach

Figure 2 shows the system architecture of WikiKreator. We are required to generate content to populate sections of the stubs (S_1, S_2, etc.) that belong to category C_1. Categories on Wikipedia group together pages on similar subjects. Hence, categories characterize Wikipedia articles surprisingly well (Zesch and Gurevych, 2007). Naturally, we leverage knowledge existing in the categories to build our text classifier. To learn category specific templates, the system should learn from articles contained within the same or similar categories. WikiKreator learns category-specific templates using all the articles that can be reached using a top-down approach from the particular category. For example, in addition to C_1, WikiKreator also learns templates from articles in C_2 and C_3 (the subcategories of C_1). As shown in the Figure 2, we deploy a two stage process to generate content for a stub:

[i] Content Retrieval and

[ii] Content Summarization.

In the first stage, our focus is to retrieve content that is relevant to the stub, say, S_1 that belongs to C_1. We extract all the articles that belong to C_1 and the subcategories, namely, C_2 and C_3. A training set is created with the contents in the sections of the articles as instances and the section titles as the corresponding classes. Topic distribution vectors for each section content are generated using LDA (Blei et al., 2003). We train a Random Forest

(RF) classifier using the topic distribution vectors. As mentioned earlier, only the top 10 most frequent sections are considered for the multi-class classification task. We retrieve relevant excerpts from the web by formulating queries. The topic model infers the topic distribution features of each excerpt and the RF classifier predicts the section (s_1, s_2, etc.) of the excerpt. All web automation tasks are performed using HTMLUnit[6]. In the second stage, our ILP based summarization approach synthesizes information from multiple excerpts assigned to a section and presents the most informative and linguistically well-formed summary as the corresponding content for each section. A word-graph is constructed that generates several sentences; only a few of the sentences are retained based on the ILP solution. The predicted section is entered in the stub article along with the final sentences selected by the ILP solution as the corresponding section-specific content on Wikipedia.

3.1 Content Retrieval

Article Extraction: Wikipedia provides an API[7] to download articles in the XML format. Given a category, the API is capable of extracting all the articles under it. We recursively extract articles by identifying all the categories in the hierarchy that can be reached by the crawler using top-down traversal. We use a simple python script[8] to extract the section titles and the corresponding text content from the XML dump.

Classification model: WikiKreator uses Latent Dirichlet Allocation (LDA) to represent each document as a vector of topic distributions. Each topic is further represented as a vector of probabilities of word distributions. Our intuition is that the topic distribution vectors of the same sections across different articles would be similar. Our objective is to learn these topic representations, such that we can accurately classify any web excerpt by inferring the topics in the text. Say C, a category on Wikipedia, has k Wikipedia articles (W).

$$(C) = \{W_1, W_2, W_3, W_4, ..., W_k\}$$

Each article W_j has several sections denoted as $s_{ji}c_{ji}$ where s_{ji} and c_{ji} refer to the section title and content of the ith section in the jth article, respectively. We concentrate on the 10 most frequent sections in any category. Training using content from sections that are not frequent might result in sub-optimal classification models. In our experiments, each frequent section had enough instances to optimally train a classifier. Let us denote the 10 most frequent sections in any category as $\boldsymbol{S}$. If any s_{ji} from W_j exists in $\boldsymbol{S}$, the content (c_{ji}) is included in the training set along with the section title (s_{ji}) as the corresponding class label. These steps are repeated for all the articles in the category. Each instance is then represented as:

$$c_{ji} = \{p_{ji}(t_1), p_{ji}(t_2), p_{ji}(t_3), \ldots, p_{ji}(t_m)\}$$

where m is the number of topics. s_{ji} is the corresponding label for this training instance. The set of topics are t_1, t_2, t_3,..., t_m while $p_{ji}(t_m)$ refers to the probability of topic m of content c_{ji}. Contents from the most frequent sections are each considered as a document and LDA is applied to generate document-topic distributions. We experiment with several values of m and use the value that generates the best classification model in each category. The topic vectors and the corresponding labels are used to train a Random Forest (RF) classifier. As the classes might be unbalanced, we apply resampling on the training set.

Predicting sections: In this step, we search the web for relevant content on the stub and assign them to their respective sections. We formulate search queries to retrieve web pages using a search engine. We extract multiple excerpts from the pages and then the RF classifier predicts the class (section label) for each excerpt.

(i) Query Generation: To search the web, we formulate queries by combining the stub title and keyphrases extracted from the first sentence of the introductory content of the stub. The first sentence generally contains the most important keywords that represent the article. Focused queries increases relevance of extraction as well as helps in disambiguation of content. We use the topia term extractor (Chatti et al., 2014) to extract keyphrases. For example, the query generated for a stub on *Hereditary hyperbilirubinemia* is *Hereditary hyperbilirubinemia bilirubin metabolic disorder* where *bilirubin metabolic disorder* are the keyphrases generated from the first sentence of the stub from Wikipedia. The query is used to identify the top 20 URLs (search results) from Google[9].

(ii) Boilerplate removal: Web content from the search results obtained in the previous step re-

[6] http://htmlunit.sourceforge.net/

[7] https://en.wikipedia.org/wiki/Special:Export

[8] http://medialab.di.unipi.it/wiki/Wikipedia_Extractor

[9] http://www.google.com

quires cleaning to retain only the relevant information. Removal of irrelevant content is done using boilerplate detection (Kohlschütter et al., 2010). The web pages contain several excerpts (text elements) in between the HTML tags. Only the excerpts that are classified as relevant by the boilerplate detection technique are retained.

(iii) Classification and assignment of excerpts: The LDA model generated earlier infers topic distribution of each excerpt based on word distributions. The RF classifier predicts the class (section title) for each excerpt based on the topic distribution. However, predictions that do not have a high level of confidence might lead to excerpts being appended to inappropriate sections. Therefore, we set the minimum confidence level at 0.5. If the prediction confidence of the RF classifier for a particular excerpt is above the minimum confidence level, the excerpt is assigned to the class; otherwise, the excerpt is discarded.

In the next step, we apply summarization on the excerpts assigned to each section.

3.2 Content Summarization

To summarize content for Wikipedia effectively, we formulate an ILP problem to generate abstractive summaries for each section with the objective of maximizing linguistic quality and information content.

Word-graph: A word-graph is constructed using all the sentences included in the excerpts assigned to a particular section. We used the same technique to construct the word-graph as (Filippova, 2010) where the nodes represent the words (along with parts-of-speech (POS)) and directed edges between the nodes are added if the words are adjacent in the input sentences. Each sentence is connected to dummy *start* and *end* nodes to mark the beginning and ending of the sentences. The sentences from the excerpts are added to the graph in an iterative fashion. Once the first sentence is added, words from the following sentences are mapped onto a node in the graph provided that they have the exact same word form and the same POS tag. Inclusion of POS information prevents ungrammatical mappings. The words are added to the graph in the following order:

- Content words are added for which there are no candidates in the existing graph;
- Content words for which multiple mappings are possible or such words that occur more than once in the sentence;
- Stopwords.

If multiple mappings are possible, the context of the word is checked using word overlaps to the left and right within a window of two words. Eventually, the word is mapped to that node that has the highest context. We also changed Filippova's method by adding punctuations as nodes to the graph. Figure 1 shows a simple example of the word-graph generation technique. We do not show POS and punctuations in the figure for the sake of clarity. The Figure also shows that several possible paths (sentences) exist between the dummy *start* and *end* nodes in the graph. Ideally, excerpts for any section would contain multiple common words as they belong to the same topic and have been assigned the same section. The presence of common words ensure that new sentences can be generated from the graph by fusing original set of sentences in the graph. Figure 1 shows an illustration of our approach where the set of sentences assigned to a particular section (WG_1) are used to create the word-graph. The word-graph generates several possible paths between the dummy nodes; we show only three such paths (WG_2). To obtain abstractive summaries, we remove generated paths from the graph that are same or very similar to any of the original sentences. If the cosine similarity of a generated path to any of the original sentences is greater than 0.8, we do not retain the path. We compute cosine similarity after applying stopword removal. However, we do not apply stemming as our graph construction is based on words existing in the same form in multiple sentences. Similar to Filippova's work, we set the minimum path length (in words) to eight to avoid incomplete sentences. Paths without verbs are discarded. The final set of generated paths after discarding the ineligible ones are used in the next step of summary generation.

3.2.1 ILP based Path Selection

Our goal is to select paths that maximize the informativeness and linguistic quality of the generated summaries. To select the best multiple possible sentences, we apply an *overgenerate and select* (Walker et al., 2001) strategy. We formulate an optimization problem that 'selects' a few of the many generated paths in between the dummy nodes from the word-graph. Let p_i denote each path obtained from the word-graph. We introduce three different factors to judge the relevance of

a path – *Local informativeness* ($I^{loc}(p_i)$), *Global informativeness* ($I^{glob}(p_i)$) and *Linguistic quality* ($LQ(p_i)$). Any sentence path should be relevant to the central topic of the article; this relevance is tackled using $I^{glob}(p_i)$. $I^{loc}(p_i)$ models the importance of a sentence among several possible sentences that are generated from the word-graph. Linguistic quality ($LQ(p_i)$) is computed using a trigram language model (Song and Croft, 1999) that assigns a logarithmic score of probabilities of occurrences of three word sequences in the sentences.

Local Informativeness: In principle, we can use any existing method that computes sentence importance to account for *Local Informativeness*. In our model, we use TextRank scores (Mihalcea and Tarau, 2004) to generate an importance value of each path. TextRank creates a graph of words from the sentences. The score of each node in the graph is calculated as shown in Equation (1):

$$S(V_i) = (1-d) + d \times \sum_{V_j \in adj(V_i)} \frac{w_{ji}}{\sum_{V_k \in adj(V_i)} w_{jk}} S(V_i) \quad (1)$$

where V_i represents the words and $adj(V_i)$ denotes the adjacent nodes of V_i. Setting d to 0.80 in our experiments provided the best content selection results. The computation convergences to return final word importance scores. The informativeness score of a path $I^{loc}(p_i)$ is obtained by adding the importance scores of the individual words in the path.

Global Informativeness: To compute global informativeness, we compute the relevance of a sentence with respect to the query to assign higher weights to sentences that explicitly mention the main title or mention certain keywords that are relevant to the article. We compute the cosine similarity using TFIDF features between each sentence and the original query that was formulated during the web search stage. We define global informativeness as follows:

$$I^{glob}(p_i) = CosineSimilarity(Q, p_i) \quad (2)$$

where Q denotes the formulated query.

Linguistic Quality: In order to compute *Linguistic quality*, we use a language model that assigns probabilities to sequence of words to compute linguistic quality. Suppose a path contains a sequence of q words $\{w_1, w_2, ..., w_q\}$. The score $LQ(p_i)$ assigned to each path is defined as follows:

$$LQ(p_i) = \frac{1}{1 - LL(w_1, w_2, ..., w_q)}, \quad (3)$$

where $LL(w_1, w_2, ..., w_q)$ is defined as:

$$LL(w_1, \ldots, w_q) = \frac{1}{L} \cdot \log_2 \prod_{t=3}^{q} P(w_t | w_{t-1} w_{t-2}). \quad (4)$$

As can be seen from Equation (4), we combine the conditional probability of different sets of 3-grams (trigrams) in the sentence and averaged the value by L – the number of conditional probabilities computed. The $LL(w_1, w_2, \ldots, w_q)$ scores are negative; with higher magnitude implying lower importance. Therefore, in Equation (3), we take the reciprocal of the logarithmic value with smoothing to compute $LQ(p_i)$. In our experiments, we used a 3-gram model[10] that is trained on the English Gigaword corpus. Trigram models have been successfully used in several text-to-text generation tasks (Clarke and Lapata, 2006; Filippova and Strube, 2008) earlier.

ILP Formulation: To select the best paths, we combine all the above mentioned factors $I^{loc}(p_i)$, $I^{glob}(p_i)$ and linguistic quality $LQ(p_i)$ in an optimization framework. We maximize the following objective function:

$$F(p_1, \ldots, p_K) = \sum_{i=1}^{K} \frac{1}{T(p_i)} \cdot I^{loc}(p_i) \cdot I^{glob}(p_i) \cdot LQ(p_i) \cdot p_i \quad (5)$$

where K represents the total number of generated paths. Each p_i represents a binary variable, that can be either 0 or 1, depending on whether the path is selected in the final summary or not. In addition, $T(p_i)$ – the number of tokens in a path, is included in the objective function. The term $\frac{1}{T(p_i)}$ normalizes the Textrank scores by the length of the sentences. First, we ensure that a maximum of S_{max} sentences are selected in the summary using Equation (6).

$$\sum_{i=1}^{K} p_i \leq S_{max} \quad (6)$$

In our experiments, we set S_{max} to 5 to generate short concise summaries in each section. Using a length constraint enables us to only populate the sections using the most informative content. We introduce Equation (7) to prevent similar information (cosine similarity ≥ 0.5) from being conveyed

[10]The model is available here: `http://www.keithv.com/software/giga/`. We used the VP 20K vocab version.

Category	Most Frequent Sections
American Mathematicians	*Awards, Awards and honors, Biography, Books, Career, Education, Life, Publications, Selected publications, Work*
Diseases and Disorders	*Causes, Diagnosis, Early life, Epidemiology, History, Pathophysiology, Prognosis, Signs and symptoms, Symptoms, Treatment*
US Software companies	*Awards, Criticism, Features, Games, History, Overview, Products, Reception, Services, Technology*

Table 1: Data characteristics of three domains on Wikipedia

Category	#Articles	#Instances
American Mathematicians	$\sim$ 2100	1493
Diseases and Disorders	$\sim$ 7000	9098
US Software companies	$\sim$ 3600	2478

Table 2: Dataset used for classification

by different sentences. This constraint reduces redundancy. If two sentences have a high degree of similarity, only one out of the two can be selected in the summary.

$$\forall i, i' \in [1, K], i \neq i', \\ p_i + p_{i'} \leq 1 \text{ if } sim(p_i, p_{i'}) \geq 0.5. \quad (7)$$

The ILP problem is solved using the Gurobi optimizer (2015). The solution to the problem decides the paths that should be included in the final summary. We populate the sections on Wikipedia using the final summaries generated for each section along with the section title. All the references that have been used to generate the sentences are appended along with the content generated on Wikipedia.

4 Experimental Results

To evaluate the effectiveness of our proposed technique, we conduct several experiments. First, we evaluate our content generation approach by generating content for comprehensive articles that already exist on Wikipedia. Second, we analyze reviewer reactions on our system generated articles by adding content to several stubs on Wikipedia. Our experiments were designed to answer the following questions:

(i)*What are the optimal number of topic distribution features for each category? What are the classification accuracies in each domain?*

(ii)*To what extent can our technique generate the content for articles automatically?*

(iii)*What are the general reviewer reactions on Wikipedia and what percentage of automatically generated content on Wikipedia is retained?*

Dataset Construction: As mentioned earlier in Section 3.1, we crawl Wikipedia articles by traversing the category graph. Articles that contain at least three sections were included in the training set; other articles having lesser number of sections

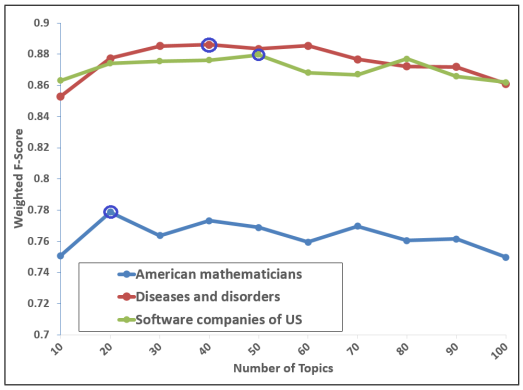

Figure 3: Performance of Classifier in the three categories based on the number of topics.

are generally labeled as stubs and hence not used for training. Table 1 shows the most frequent sections in each category. Further, Table 2 shows the total number of articles retrieved from Wikipedia in each category. The total number of instances are also shown. The number of instances denotes the total number of the most frequent sections in each category. As can be seen from the table, the number of instances is higher than the number of articles only in case of the category on *diseases*. This implies that there are generally more common sections in the diseases category than the other categories.

In each category, the content from only the most frequent sections were used to generate a topic model. The topic model is further used to infer topic distribution vectors from the training instances. We used the MALLET toolkit (McCallum, 2002) for generating topic distribution vectors and the WEKA package (Hall et al., 2009) for the classification tasks.

Optimal number of topics: The LDA model requires a pre-defined number of topics. We experiment with several values of the number of topics ranging from 10 to 100. The topic distribution features of the content of the instances are used to train a Random Forest Classifier with the corresponding section titles as the class labels. As can be seen in the Figure 3, the classification performance varies across domains as well as on the number of topics. The optimal number of topics based on the dataset are marked in *blue* cir-

Category	LDA-RF	SVM-WV
American Mathematicians	**0.778**	0.478
Diseases and Disorders	**0.886**	0.801
US Software companies	**0.880**	0.537

Table 3: Classification: Weighted F-Scores

cles (40, 50 and 20 topics for *Diseases*, *Software Companies in US* and *American mathematicians*, respectively) in the Figure. We classify web excerpts using the best performing classifiers trained using the optimal number of topic features in each category.

Classification performance: We use 10-fold cross validation to evaluate the accuracy of our classifier. According to the F-Scores, our classifier **(LDA-RF)** performs similarly in the categories on *Diseases* and *US Software companies*. However, the accuracy is lower in the *American Mathematicians* category. We also experimented with a baseline classifier, that is trained on TFIDF features (upto trigrams). A Support vector machine (Cortes and Vapnik, 1995) classifier obtained the best performance using the TFIDF features. The baseline system is referred to as **SVM-WV**. We experimented with several other combinations of classifiers; however, we show only the best performing systems using the LDA and TFIDF features. As can be seen from the Table 3, our classifier **(LDA-RF)** outperforms **SVM-WV** significantly in all the domains. **SVM-WV** performs better in the category on diseases than the other two categories and the performance is comparable to **(LDA-RF)**. The diseases category has more uniformity in terms of the section titles, hence specific words or phrases characterize the sections well. In contrast, word distributions (LDA) work significantly better than TFIDF features in the other two categories.

Error Analysis: We performed error analysis to understand the reason for misclassifications. As can be seen from the Table 1, all the categories have several overlapping sections. For example, *Awards and honors* and *Awards* contain similar content. Authors use various section names for similar content in the articles within the same category. We analyzed the confusion matrices, and found that multiple instances in *Awards* were classified into the class of *Awards and honors*. Similar observations are made on the *Books* and *Publications* classes – which are related sections in the context of academic biographies. In future, we plan to use semantic measures to relate similar classes automatically and group them in the same class during classification.

Category	System	ROUGE-1	ROUGE-2
American Mathematicians	WikiKreator	**0.522**	**0.311**
	Perceptron	0.431	0.193
	Extractive	0.471	0.254
Diseases and Disorders	WikiKreator	**0.537**	**0.323**
	Perceptron	0.411	0.197
	Extractive	0.473	0.232
US Software companies	WikiKreator	**0.521**	**0.321**
	Perceptron	0.421	0.228
	Extractive	0.484	0.257

Table 4: ROUGE-1 and 2 Recall values – Comparing system generated articles to model articles

Content Selection Evaluation: To evaluate the effectiveness of our content generation process, we generated the content of 500 randomly selected articles that already exist on Wikipedia in each of the categories. We compare WikiKreator's output against the current content of those articles on Wikipedia using ROUGE (Lin, 2004). ROUGE matches N-gram sequences that exist in both the system generated articles and the original Wikipedia articles (gold standard). We also compare WikiKreator's output with an existing Wikipedia generation system [**Perceptron**] of Sauper and Barzilay (2009)[11] that employs a perceptron learning framework to learn topic specific extractors. Queries devised using the conjunction of the document title and the section title were used to obtain excerpts from the web using a search engine, which were used in the perceptron model. In *Perceptron*, the most important sections in the category was determined using a bisectioning algorithm to identify clusters of similar sections. To understand the effectiveness of our abstractive summarizer, we design a system (**Extractive**) that uses an extractive summarization module. In *Extractive*, we use LexRank (Erkan and Radev, 2004) as the summarizer instead of our ILP based abstractive summarization model. We restrict the extractive summaries to 5 sentences for accurate comparison of both the systems. The same content was received as input from the classifier by the *Extractive* as well as our ILP-based system.

As can be seen from the Table 4, the ROUGE scores obtained by WikiKreator is higher than that of the other comparable systems in all the categories. The higher ROUGE scores imply that *WikiKreator* is generally able to retrieve useful information from the web, synthesize them and present the important information in the article.

[11]The system is available here: `https://github.com/csauper/wikipedia`

Statistics	Count
Number of stubs edited	40
Number of stubs retained without any changes	21
Number of stubs that required minor editing	6
Number of stubs where edits were modified by reviewers	4
Number of stubs in which content was removed	9
Average change in size of stubs	515 bytes
Average number of edits made post content-addition	∼3

Table 5: Statistics of Wikipedia generation

However, it may also be noted that the *Extractive* system outperforms the Perceptron framework. Summarization from multiple sources generates more informative summaries and is more effective than 'selection' of the most informative excerpt, which is often inadequate due to potential loss of information. WikiKreator performs better than the extractive system on all the categories. Our ILP-based abstractive summarization system fuses and selects content from multiple sentences, thereby aggregating information successfully from multiple sources. In contrast, LexRank 'extracts' the top 5 sentences that results in some information loss.

Analysis of Wikipedia Reviews: To compare our method with the other techniques, it is necessary to generate content and append to Wikipedia stubs using all the techniques. However, recent work on article generation (Banerjee et al., 2014) has already shown that content directly copied from web sources cannot be used on Wikipedia. Further, bots using copyrighted content might be banned and real-users would have to read sub-standard articles due to the internal tests we perform. Due to the above mentioned reasons, we appended content generated only using our abstractive summarization technique.

We published content generated by WikiKreator on Wikipedia and appended the content to 40 randomly selected stubs. As can be seen from the Table 5, the content generated using our system was generally accepted by the reviewers. Half of the articles did not require any further changes; while in 6 cases (15%) the reviewers asked us to fix grammatical issues. In 9 stubs, the reliability of the cited references was questioned. Information sources on Wikipedia need to satisfy a minimum reliability standard, which our algorithm currently cannot determine. On an average, 3 edits were made to the Wikipedia articles that we generated. In general, there is an average increase in the content size of the stubs that we edited showing that our method is capable of producing content that generally satisfy Wikipedia criterion.

Analysis of section assignment: We manually inspected generated content of 20 articles in each category. Generated summaries are both informative and precise. However, in certain cases, the generated section title is not the same as the section title in the original Wikipedia article. For example, we generated content for the section "Causes" for the article on Middle East Respiratory Syndrome (MERS)[12]:

Milk or meat may play a role in the transmission of the virus . People should avoid drinking raw camel milk or meat that has not been properly cooked . There is growing evidence that contact with live camels or meat is causing MERS.

The corresponding content on the Wikipedia is in a section labeled as "Transmission". Section titles at the topmost level in a category might not be relevant to all the articles. Instead of using a top-down approach of traversing the category-graph, we can also use a bottom-up approach where we learn from all the categories that an article belongs to. For example, the article on MERS belongs to two categories: *Viral respiratory tract infection* and *Zoonoses*. Training using all the categories will allow context-driven section identification. Most frequent sections at a higher level in the category graph might not always be relevant to all the articles within a category.

5 Conclusions and Future Work

In this work, we presented WikiKreator that can generate content automatically to improve Wikipedia stubs. Our technique employes a topic-model based text classifier that assigns web excerpts into various sections on an article. The excerpts are summarized using a novel abstractive summarization technique that maximizes informativeness and linguistic quality of the generated summary. Our experiments reveal that WikiKreator is capable of generating well-formed informative content. The summarization step ensures that we avoid any copyright violation issues. The ILP based sentence generation strategy ensures that we generate novel content by synthesizing information from multiple sources and thereby improve content selection. In future, we plan to cluster related sections using semantic relatedness measures. We also plan to estimate reliabilities of sources to retrieve information only from reliable sources.

[12] https://en.wikipedia.org/wiki/Middle_East_respiratory_syndrome

References

Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. 1999. *Modern information retrieval*, volume 463. ACM press New York.

Siddhartha Banerjee, Cornelia Caragea, and Prasenjit Mitra. 2014. Playscript classification and automatic wikipedia play articles generation. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 3630–3635, Aug.

David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022.

Mohamed Amine Chatti, Darko Dugoija, Hendrik Thus, and Ulrik Schroeder. 2014. Learner modeling in academic networks. In *Advanced Learning Technologies (ICALT), 2014 IEEE 14th International Conference on*, pages 117–121. IEEE.

James Clarke and Mirella Lapata. 2006. Constraint-based sentence compression an integer programming approach. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 144–151. Association for Computational Linguistics.

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning*, 20(3):273–297.

Vipul Dalal and Latesh Malik. 2013. A survey of extractive and abstractive text summarization techniques. In *Emerging Trends in Engineering and Technology (ICETET), 2013 6th International Conference on*, pages 109–110. IEEE.

Günes Erkan and Dragomir R Radev. 2004. Lexrank: Graph-based lexical centrality as salience in text summarization. *J. Artif. Intell. Res.(JAIR)*, 22(1):457–479.

Katja Filippova and Michael Strube. 2008. Sentence fusion via dependency graph compression. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 177–185. Association for Computational Linguistics.

Katja Filippova. 2010. Multi-sentence compression: Finding shortest paths in word graphs. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 322–330. Association for Computational Linguistics.

Evgeniy Gabrilovich and Shaul Markovitch. 2007. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *IJCAI*, volume 7, pages 1606–1611.

Juri Ganitkevitch, Chris Callison-Burch, Courtney Napoles, and Benjamin Van Durme. 2011. Learning sentential paraphrases from bilingual parallel corpora for text-to-text generation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1168–1179. Association for Computational Linguistics.

Shima Gerani, Yashar Mehdad, Giuseppe Carenini, T. Raymond Ng, and Bita Nejat. 2014. Abstractive summarization of product reviews using discourse structure. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1602–1613. Association for Computational Linguistics.

Inc. Gurobi Optimization. 2015. Gurobi optimizer reference manual.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. 2009. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18.

Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. 2010. Boilerplate detection using shallow text features. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 441–450. ACM.

Peng Li, Yinglin Wang, and Jing Jiang. 2013. Automatically building templates for entity summary construction. *Information Processing & Management*, 49(1):330–340.

Andy Liaw and Matthew Wiener. 2002. Classification and regression by randomforest. *R news*, 2(3):18–22.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81.

Andrew K McCallum. 2002. {MALLET: A Machine Learning for Language Toolkit}.

Olena Medelyan, Ian H Witten, and David Milne. 2008. Topic indexing with wikipedia. In *Proceedings of the AAAI WikiAI workshop*, pages 19–24.

Rada Mihalcea and Paul Tarau. 2004. Textrank: Bringing order into texts. Association for Computational Linguistics.

Ani Nenkova, Sameer Maskey, and Yang Liu. 2011. Automatic summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts of ACL 2011*, page 3. Association for Computational Linguistics.

Christina Sauper and Regina Barzilay. 2009. Automatically generating wikipedia articles: A structure-aware approach. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 208–216. Association for Computational Linguistics.

Fei Song and W Bruce Croft. 1999. A general language model for information retrieval. In *Proceedings of the eighth international conference on Information and knowledge management*, pages 316–321. ACM.

Wei Song, Yu Zhang, Ting Liu, and Sheng Li. 2010. Bridging topic modeling and personalized search. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 1167–1175. Association for Computational Linguistics.

Marilyn A Walker, Owen Rambow, and Monica Rogati. 2001. Spot: A trainable sentence planner. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, pages 1–8. Association for Computational Linguistics.

Conglei Yao, Xu Jia, Sicong Shou, Shicong Feng, Feng Zhou, and HongYan Liu. 2011. Autopedia: automatic domain-independent wikipedia article generation. In *Proceedings of the 20th international conference companion on World wide web*, pages 161–162. ACM.

Torsten Zesch and Iryna Gurevych. 2007. Analysis of the wikipedia category graph for nlp applications. In *Proceedings of the TextGraphs-2 Workshop (NAACL-HLT 2007)*, pages 1–8.