Classifying and Searching News Articles from Key Phrases

Mehmet Acikgoz
School of Computing and Engineering
University of Missouri - Kansas City
Kansas City, Missouri, USA
ma96f@mail.umkc.edu

Irfan Ahmed
School of Computing and Engineering
University of Missouri - Kansas City
Kansas City, Missouri, USA
iar2f@mail.umkc.edu

Jonathan Wolfe
School of Computing and Engineering
University of Missouri - Kansas City
Kansas City, Missouri, USA
jawhf4@mail.umkc.edu

Abstract - One of the most common tasks of users of news publication sites is finding articles relating to a specific topic. We present an application that can automatically extract relevant key features from the text of the article and classify the article into the correct category for users to be able to quickly find a desired article. We used the Rapid Automatic Keyword Extraction algorithm to extract key phrases, Multinomial Logistic Regression to classify articles and RankNet to search for articles. We found that these systems provided moderate success in providing the desired information and with further adjustments to each area the system will automatically deliver reliable information and search results.

I. INTRODUCTION

Information Retrieval (IR) is the act of searching for information in a document, a document itself or metadata about a document. One of the most commonly used applications of IR is a search engine. These applications utilize various features of data to present the most relevant results to the user. These features are generally extracted from the data and may also have further processing done to provide further insight into presenting the data to the user. In this paper, we will be presenting a method of extracting key phrases from articles from Huffington Post. Then we further process these key phrases into categories. Finally, we present a very simple search engine that uses these extracted key phrases and classifications of categories to present users the most relevant articles to their search query.

II. RELATED WORKS

Keyword extraction is a subfield of IR which is about gathering words and phrases from text documents. There are several techniques and algorithms used for keywords extraction such as TF-IDF, TextRank and Rake. TF-IDF stands for Text Frequency Inverse Document Frequency. TF-IDF measures how important a word is to a document in a collection of documents [1],[2]. TextRank algorithm is based on PageRank. It creates a graph of the words and relationships between them from a document, then identifies the most important words of the graph based on importance scores calculated recursively from the entire graph [1],[2]. While these techniques are handy only for word extraction, Rake is also good at finding related words and phrases which provide information about semantics and summary of the article.

In IR, classification of text can be used to gain new insight into the data being classified or group the data into predetermined groups for better referencing. Referencing data by groups can provide a faster and more accurate means of finding important information. There are multiple different models that can achieve classification, three of importance are Logistic Regression, Naive Bayes and Random Forest Classifier. Logistic Regression and Naive Bayes have been found in [3] to deliver desirable results and are utilized here to find an accurate model for classification along with Random Forest Classifier.

IR is the underlying science behind the functionality of search engines. One of the fundamental problems for developing a search engine is ranking documents in an order desirable to the user. A technique for solving this ranking problem uses supervised machine learning to find an optimal relative ordering of items is Learning to Rank. In 2005, Chris Burges, et al. [4] with Microsoft developed RankNet, one of the most influential Learning to Rank algorithms. This algorithm was developed using neural nets and a cost function to minimize the occurrence of incorrect pair ordering using Stochastic Gradient Descent. The algorithm takes a pair of document features and returns the probability that the first document is ranked higher than the second. This algorithm is implemented in this project to deliver a ranked list of articles relating to a search query in a desirable order.

III. ARCHITECTURE AND METHODOLOGY

A. Key Phrase Extraction

The dataset we used in this project contains more than 200,000 records about news articles from Huffpost between the years 2012 and 2018 from [5]. Each record contains information about the category, headline, date, authors, hyperlink, short description, and date of the article. The dataset has around 42 classifications of categories. The dataset was sampled to have an equal number of articles in each category to create a balanced dataset and reduce the computation burden in future article classification and searching. A 'keyphrases' set was added to the dataset, by extracting out meaningful phrases and compressing each article into a better format for processing in classification and searching.

Rapid Automatic Keyword Extraction (RAKE) algorithm is a popular keyword extraction technique, which is available in NLTK and some other libraries. It is used to find keyword sequences which contain uncommon words that usually appear together [8]. RAKE assumes keywords do not include stop words, are infrequent and frequently appear together. It scores words using formula 1:

$$\text{Score (word)} = (\text{deg(word)} / \text{frequency(word)}) \tag{1}$$

The score of a word is a ratio of its degree to its frequency. Words with higher scores are those which are not frequent in the document and co-occurs with other words frequently. The frequency of a word is how often the word occurs in the document. Since frequency is the denominator of the equation, the higher frequencies cause lower scores. The degree of a word is a measure of how often a word co-occurs with other words. The degree of a word centers on the sum of the co-occurrences of the word with other words in the document [7].

For preprocessing of the dataset, we accessed the article with its hyperlink and scraped the main text of the article by using the beautiful soup library as explained in [6]. Next, we derived the key phrases for each article by implementing RAKE and saved up to 20 key phrases from each article and stored them as a new set in the dataset.

The accuracy of article classification and searching depends on the accuracy of the key phrase extraction. Several studies have shown the accuracy of RAKE in keyword extraction [1], [2], [7]. We selected RAKE because it has shown to be a better keyword extraction algorithm in accuracy, performance, simplicity and efficiency when compared to other algorithms [9].

B. Article Classification

Once key phrases were extracted from each document, this information combined with the article classifications from the original dataset were utilized to develop a model to predict future article classifications given a set of key phrases. To utilize the key phrases, the text was converted to a matrix of word count tokens with stop words removed and words converted to lowercase before assembling the matrix. The matrix was then run through TF-IDF as represented in formula 2 and described in [10] to provide a weight to words that occur more frequently in a given article while also reducing very common words among all of the documents to emphasize the weight of a frequently used word in a document that is not common in others.

$$W_{ij} = tf_{ij} \ x \ idf = tf_i \ x \log\left(\frac{N}{n_i} + 0.001\right) \tag{2}$$

TF refers to the feature items that occur in the document, and IDF is the corrective measure of term frequency.

Three different models were assessed to determine which would yield the best results: multinomial logistic regression, multinomial Naive Bayes and Random Forest Classifier. The multinomial logistic regression model is expressed in formula 3 as described in [11]:

$$\Pr(Y_{ik}) = \frac{exp(\beta_{0k} + \boldsymbol{x_i}\boldsymbol{\beta'_k})}{\sum_{j=1}^{m} exp(\beta_{0j} + \boldsymbol{x_i}\boldsymbol{\beta'_j})} \tag{3}$$

where, *Pr(*Y_{ik}*)*, is the probability that a given observation, *i*, will belong to a category, *k*, of *m*th categories and $\boldsymbol{\beta_k}$ being the row row vector of regression coefficients of $\boldsymbol{X}$. The multinomial Naive Bayes model is expressed in formula 4 as described in [12]:

$$P(Ci|e) = \frac{P(e|Ci)P(ci)}{\sum_{!=1}^{k} P(e|cj)P(cj)} \ldots, (i = 1,2 \ldots n) \tag{4}$$

where Ci is the previous probability class; the posterior probability P (Ci|e) is obtained from the Bayesian formula.The Random Forest Classifier is expressed in formula 5 as described in [13]:

$$Entropy = -plog_2(p) - qlog_2(q) \tag{5}$$

where -p and -q is the proportion of negative samples, and p+ and q+ is the proportion of positive samples. Entropy of one means the class labels are equally divided, and zero means the sample is completely homogenous.

These models were implemented in python under the guidance of [14]. For the final analysis, a classification report was generated to observe the accuracy of each category.

C. *Article Search*

Once the documents of the dataset were classified and the key phrases were extracted, this information was stored and later used in a search function. The search function finds all related documents by a search query with related documents having at least one of the words in the query appearing in its associated key phrases. The related documents are then sorted based on three query-independent and four query-dependent features. The sorting was performed using RankNet on these seven features to establish an optimal order with the most relevant result being the highest ranked and lowest being loosely related.

The query-independent features include the count of key phrases, word count in all key phrases and age of the article in years. The key phrases list provided by the extraction phase performed earlier consists of up to twenty strings of words closely related to the document. The count of key phrases is merely a count of these strings of words with a maximum of twenty. Each *i*th phrase, p_i, in the key phrases, consists of a sequence of words, thus the cardinality of each phrase is summed together to determine the word count in all key phrases, C_w, as shown in the formula 6:

$$C_w = \sum_i |p_i| \tag{6}$$

The age of the article is simply the difference between the current year and the year of posting for the article. These features are the same regardless of the query thus they can be calculated before the search query is entered to reduce the amount of calculations that will need to be performed after the search is initiated. All the remaining features to calculate are thus dependent on the query.

To determine query-dependent features, an inverse index consisting of a set of all words contained in all documents was constructed. Each word was further associated with the set of documents the word appears in with the positions of the word in the document's key phrases list. The inverse index only needed to be constructed once and was done before the query was generated. This index allows for quick reference in associating a word to the positions it appears in every document and is required to find query-dependent features.

The query-depend features include the count of words appearing in the key phrases from the query, percent of the query words appearing in the key phrases, count of complete matches to the query appearing in the key phrases and word count of the query. For each *i*th word in the query, q_i, to the last *n*th word, the sum of each word in each phrase, w_{jk}, matching q_i to the last pth word in each phrase and last mth phrase for the document is summed with these sums for each phrase and the sum of the phrases are then summed for each word in the query to determine the raw count of words appearing in the key phrases, C_{tf}, from the query as shown in formula 7:

$$C_{tf} = \sum_{i=1}^{n}\left(\sum_{j=1}^{m}\left(\sum_{k=1}^{p}[w_{jk} = q_i]\right)\right) \tag{7}$$

For each document, d, containing a word from the query, q, in the key phrases, the ith words in the query, q_i, is summed with the result of the function $f(q_i, d)$ returning 1 if the query word, q_i, appears in the key phrases list and 0 if it does not appear in the key phrase. This summation is then divided by the cardinality of the words in the query to determine the percent of the query words appearing in the key phrases, P_m, as shown in formula 8:

$$P_m = \sum_{i=1}^{n} f(q_i, d) \Big/ |q| \tag{8}$$

For each position, p_i, of the first word in the query, q, also in the document, d, to the nth position for the word, the function, $f(p_i, q, d)$, returns 1 if the phrase is in the key phrases for the document or 0 if it does not appear in order or entirety. The sum of the complete matches, C_{cm}, for the query in the key phrases list for each document is determined and is represented by formula 9:

$$C_{cm} = \sum_{i=1}^{n} f(p_i, q, d) \tag{9}$$

The cardinality of the words in the query is computed to determine the word count of the query. These features will then be used by a neural network to determine the ranking of documents to present the most relevant document first.

Once the features were established for each document, the RankNet model could be used to order the documents based on relevance of the document to the search query. To train the model, an initial dataset was constructed using 200 document results with each document's features as previously described on queries using term frequency to order the documents. A relevancy score of 0 to 4 was then manually assigned to each document depending on how well the document reflected its query with 0 indicating no relation to the query and 4 being very closely related. The relevancy scores were thus the dependent variable and the other features were independent variables for training the model.

The RankNet model as described in [4] takes pairwise samples [A, B] in $\mathbb{R}^d$ with an associated probability that A is a higher rank than B. Each sample uses a base model to estimate a score that is subsequently used in a Siamese-like net to estimate the probability that A ranks higher than B by using a cross-entropy cost function on the difference between these scores and outputting the probability that A is ranked higher than B. This model is then used to sort documents by rank fitted on the training features. The model compares new documents with their own features by passing in two document feature sets to the model and providing the probability that the first document ranks higher than the second. This ranked list is then used to manually assign relevancy scores to each document. The model was implemented in python as demonstrated in [15].

IV. RESULTS AND DISCUSSION

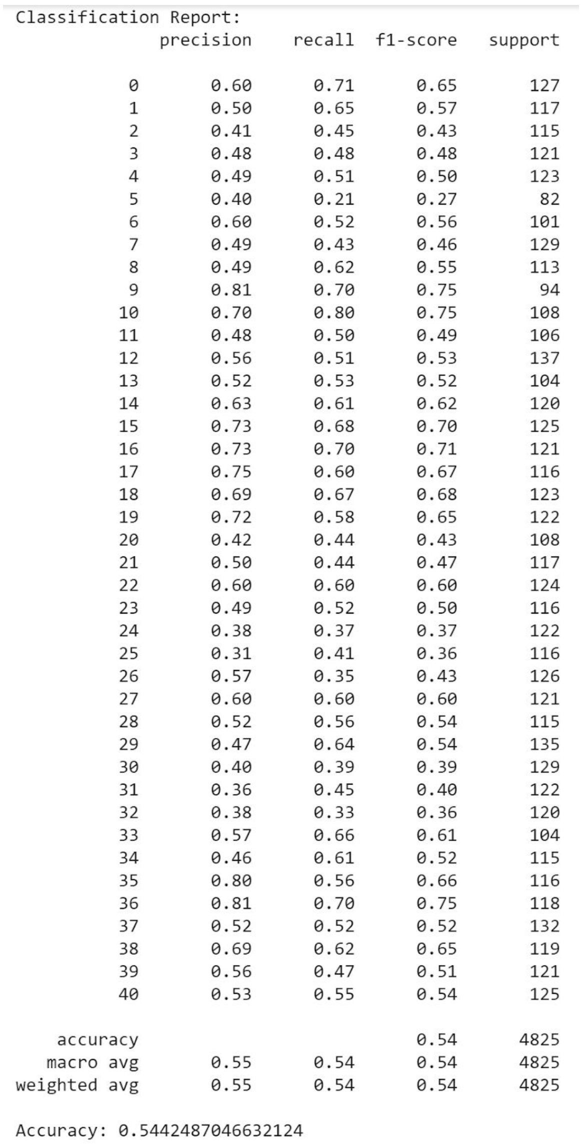
A. Key Phrase Extraction

RAKE is used in gathering the key phrases from the original articles and adding them to the dataset. The key phrases are important inputs in the later classification and searching and the accuracy of these key phrases heavily influence their results. Although some studies show that RAKE seems to be the best algorithm among TF-IDF and TextRank, several other algorithms should be tested and compared with the results of RAKE in the future to potentially improve the accuracy of key phrase extraction and subsequently article classification and searching.

B. Article Classification

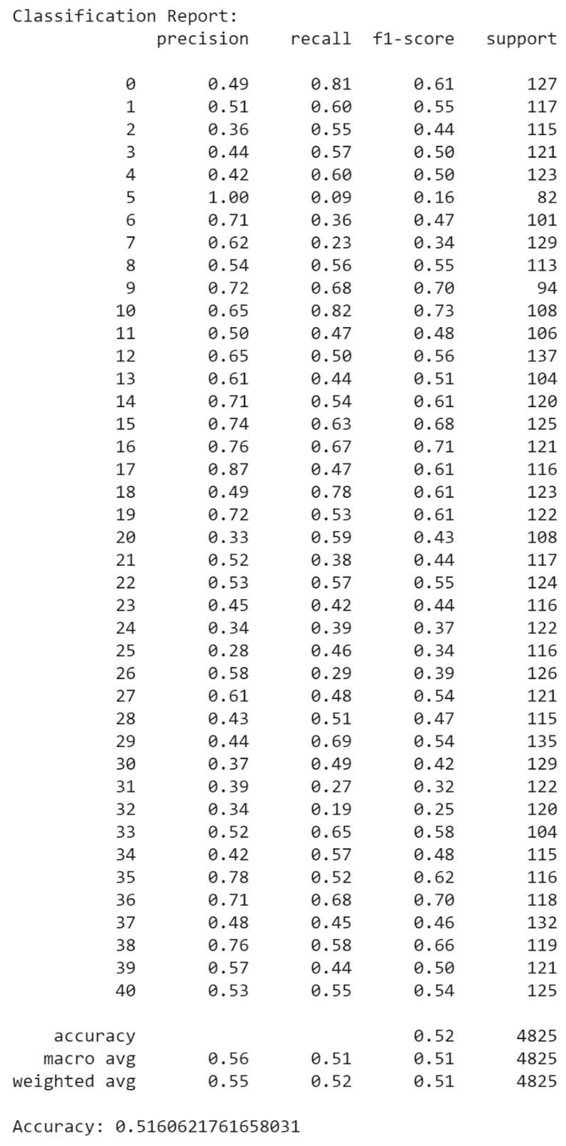
The three models used were for comparative purpose to determine which yielded the best accuracy. Of the three, logistic regression yielded the best results, with accuracy of 0.5442, second was Multinomial Naive Bayes at 0.5161 and finally Random Forest Classifier with 0.4528. For each of the models, the accuracy can be improved by increasing the sample size. Few things to consider when comparing each of the models is that logistic regression tends to show high bias and thus overfitting. This could explain the slight difference of accuracy. Table I, II and II shows the classification report for each of the models.

Table 1: Classification report for logistic regression



```
Classification Report:
              precision    recall  f1-score   support

           0       0.60      0.71      0.65       127
           1       0.50      0.65      0.57       117
           2       0.41      0.45      0.43       115
           3       0.48      0.48      0.48       121
           4       0.49      0.51      0.50       123
           5       0.40      0.21      0.27        82
           6       0.60      0.52      0.56       101
           7       0.49      0.43      0.46       129
           8       0.49      0.62      0.55       113
           9       0.81      0.70      0.75        94
          10       0.70      0.80      0.75       108
          11       0.48      0.50      0.49       106
          12       0.56      0.51      0.53       137
          13       0.52      0.53      0.52       104
          14       0.63      0.61      0.62       120
          15       0.73      0.68      0.70       125
          16       0.73      0.70      0.71       121
          17       0.75      0.60      0.67       116
          18       0.69      0.67      0.68       123
          19       0.72      0.58      0.65       122
          20       0.42      0.44      0.43       108
          21       0.50      0.44      0.47       117
          22       0.60      0.60      0.60       124
          23       0.49      0.52      0.50       116
          24       0.38      0.37      0.37       122
          25       0.31      0.41      0.36       116
          26       0.57      0.35      0.43       126
          27       0.60      0.60      0.60       121
          28       0.52      0.56      0.54       115
          29       0.47      0.64      0.54       135
          30       0.40      0.39      0.39       129
          31       0.36      0.45      0.40       122
          32       0.38      0.33      0.36       120
          33       0.57      0.66      0.61       104
          34       0.46      0.61      0.52       115
          35       0.80      0.56      0.66       116
          36       0.81      0.70      0.75       118
          37       0.52      0.52      0.52       132
          38       0.69      0.62      0.65       119
          39       0.56      0.47      0.51       121
          40       0.53      0.55      0.54       125

    accuracy                           0.54      4825
   macro avg       0.55      0.54      0.54      4825
weighted avg       0.55      0.54      0.54      4825

Accuracy: 0.5442487046632124
```

Table 2: Classification report for multinomial Naive Bayes



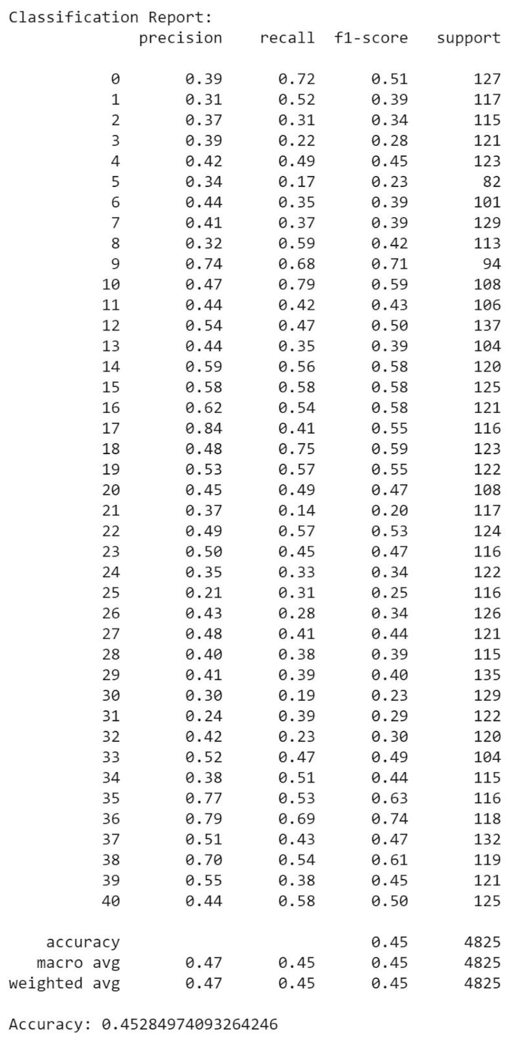
```
Classification Report:
              precision    recall  f1-score   support

           0       0.49      0.81      0.61       127
           1       0.51      0.60      0.55       117
           2       0.36      0.55      0.44       115
           3       0.44      0.57      0.50       121
           4       0.42      0.60      0.50       123
           5       1.00      0.09      0.16        82
           6       0.71      0.36      0.47       101
           7       0.62      0.23      0.34       129
           8       0.54      0.56      0.55       113
           9       0.72      0.68      0.70        94
          10       0.65      0.82      0.73       108
          11       0.50      0.47      0.48       106
          12       0.65      0.50      0.56       137
          13       0.61      0.44      0.51       104
          14       0.71      0.54      0.61       120
          15       0.74      0.63      0.68       125
          16       0.76      0.67      0.71       121
          17       0.87      0.47      0.61       116
          18       0.49      0.78      0.61       123
          19       0.72      0.53      0.61       122
          20       0.33      0.59      0.43       108
          21       0.52      0.38      0.44       117
          22       0.53      0.57      0.55       124
          23       0.45      0.42      0.44       116
          24       0.34      0.39      0.37       122
          25       0.28      0.46      0.34       116
          26       0.58      0.29      0.39       126
          27       0.61      0.48      0.54       121
          28       0.43      0.51      0.47       115
          29       0.44      0.69      0.54       135
          30       0.37      0.49      0.42       129
          31       0.39      0.27      0.32       122
          32       0.34      0.19      0.25       120
          33       0.52      0.65      0.58       104
          34       0.42      0.57      0.48       115
          35       0.78      0.52      0.62       116
          36       0.71      0.68      0.70       118
          37       0.48      0.45      0.46       132
          38       0.76      0.58      0.66       119
          39       0.57      0.44      0.50       121
          40       0.53      0.55      0.54       125

    accuracy                           0.52      4825
   macro avg       0.56      0.51      0.51      4825
weighted avg       0.55      0.52      0.51      4825

Accuracy: 0.5160621761658031
```

The model could be improved by using a larger key phrase index to determine if accuracy would improve for Multinomial Logistic Regression, Random Forest Classifier, or Multinomial Naive Bayes. In addition, using unique key words frequencies from each article could help in identifying the categories. Finally, a neural network model could be used to determine if that would be a better option than a simple classification model.

C. Article Search

To evaluate the ranking of a list of search results, Normalized Discounted Cumulative Gain (NDCG) was used as explained in [16]. NDCG was calculated by first determining the Discounted Cumulative Gain (DCG) on a list of relevancy scores ordered by the model with the first item intending to be most relevant and the last being least. Next, this list was

Table 3: Classification report for Random Forest



```
Classification Report:
              precision    recall  f1-score   support

           0       0.39      0.72      0.51       127
           1       0.31      0.52      0.39       117
           2       0.37      0.31      0.34       115
           3       0.39      0.22      0.28       121
           4       0.42      0.49      0.45       123
           5       0.34      0.17      0.23        82
           6       0.44      0.35      0.39       101
           7       0.41      0.37      0.39       129
           8       0.32      0.59      0.42       113
           9       0.74      0.68      0.71        94
          10       0.47      0.79      0.59       108
          11       0.44      0.42      0.43       106
          12       0.54      0.47      0.50       137
          13       0.44      0.35      0.39       104
          14       0.59      0.56      0.58       120
          15       0.58      0.58      0.58       125
          16       0.62      0.54      0.58       121
          17       0.84      0.41      0.55       116
          18       0.48      0.75      0.59       123
          19       0.53      0.57      0.55       122
          20       0.45      0.49      0.47       108
          21       0.37      0.14      0.20       117
          22       0.49      0.57      0.53       124
          23       0.50      0.45      0.47       116
          24       0.35      0.33      0.34       122
          25       0.21      0.31      0.25       116
          26       0.43      0.28      0.34       126
          27       0.48      0.41      0.44       121
          28       0.40      0.38      0.39       115
          29       0.41      0.39      0.40       135
          30       0.30      0.19      0.23       129
          31       0.24      0.39      0.29       122
          32       0.42      0.23      0.30       120
          33       0.52      0.47      0.49       104
          34       0.38      0.51      0.44       115
          35       0.77      0.53      0.63       116
          36       0.79      0.69      0.74       118
          37       0.51      0.43      0.47       132
          38       0.70      0.54      0.61       119
          39       0.55      0.38      0.45       121
          40       0.44      0.58      0.50       125

    accuracy                           0.45      4825
   macro avg       0.47      0.45      0.45      4825
weighted avg       0.47      0.45      0.45      4825

Accuracy: 0.45284974093264246
```

ordered by the reported relevancy score to represent an ideal ranking and its DCG was determined to give the Ideal Discounted Cumulative Gain (IDCG). NDCG is then calculated by dividing the DCG by IDCG as shown in formula 10:

$$NDCG = \frac{DCG}{IDCG} \tag{10}$$

The DCG was calculated by summing the ith relevancy score, *rel_i*, for each document to the nth document in the list and dividing it by the log of its position in the list plus 2 as shown in formula 11:

$$DCG = \sum_{i=0}^{n} \frac{rel_i}{\log_2(i+2)} \tag{11}$$

This measure gives a good indication of how well the model is ranking documents for a search query, however, assigning relevancy scores is labor intensive thus using real-world results is limited. With a limited size of the search query currently being 20 documents, the accuracy of NDCG is also limited. Two successive queries were run with each having 20 results and a new model being trained on the previous results. The first run gave an NDCG score of 0.85 and the second gave 0.57 indicating more difficulty in finding optimal ranks.

While the results of the search model are promising, the data in which to train the model was limited, thus the accuracy of the model in delivering the most relevant results was further limited. With more query data and a larger array of features for each document, the model would undoubtedly perform with better accuracy. An observed issue was that the model did not have a way to discern words with a significant meaning to the query as opposed to words that had little meaning. Features with the ability to give more meaning to the sort would greatly increase accurate ranking. Another issue is that the model currently only includes articles with at least one word from the query in the key phrases list. While this can reduce the amount of unrelated articles from appearing in the results, it might also exclude articles that are highly relevant while also including articles that are not related. An additional problem was observed while running the model, while the features were all calculated quickly, the sorting of the search results took longer than expected, which is undesirable for a set of results that are expected to be returned within a second.

V. CONCLUSION

We have seen modest results from a system with limited input for training. The search engine was able to return some of the more desirable results to the top of a list even though it was only trained on 200 ratings with a limited set of features. The classifier was able to accurately assign the category to an article nearly half of the time and while that may seem low, it is properly selecting from 42 categories. The key phrase extraction was able to generate a list of phrases that closely resemble the article from which they came. These are all very promising results for the limited datasets that were utilized. With more data and potentially implementing some of the methods previously mentioned, this system will improve greatly.

VI. REFERENCES

[1] X. LIANG. "*Understand TextRank for Keyword Extraction by Python:A scratch implementation by Python and spaCy to help you understand PageRank and TextRank for Keyword Extraction.*" Towards Data Science. https://towardsdatascience.com /textrank-for-keyword-extraction-by-python-c0bae21bcec0#:~:text=TextRank%20is%20an%20algorithm%20based,Extraction%20with%20TextRank%2C%20NER%2C%20etc. (accessed Nov. 29, 2020).

[2] "*Keyword Extraction*." MonkeyLearn. https://monkeylearn.com/keyword-extraction/. (accessed Nov. 29, 2020).

[3] M. Y. Helmi Setyawan, R. M. Awangga and S. R. Efendi, "*Comparison Of Multinomial Naive Bayes Algorithm And Logistic Regression For Intent Classification In Chatbot*," in *Intl. Conf. Appl. Engr.*, IEEE, Batam, Indonesia, Oct. 3-4, 2018, pp. 1-5, doi: 10.1109/INCAE.2018.8579372.

[4] C. Burges, et al., "*Learning to rank using gradient descent*", in *Proc. 22nd Intl. Conf. Machine Learning*, ACM, Bonn, Germany, Aug. 7-11, 2005, pp. 89–96, doi: 10.1145/1102351.1102363.

[5] R. Misra. "*News Category Dataset: Identify the type of news based on headlines and short descriptions*." Kaggle. https://www.kaggle.com/ rmisra/news-category-dataset. (accessed Nov. 29, 2020).

[6] L. Richardson. "*Beautiful Soup 4.9.0 documentation*." Crummy. https://www.crummy.com/ software/BeautifulSoup/bs4/doc/. (accessed Nov. 29, 2020).

[7] V. B. Sharma. "*rake-nltk.*" Github. https://csurfer.github.io/rake-nltk/_build/html/index.html. (accessed Nov. 29, 2020).

[8] J. Ravi. (2019). "*Mining Data from Text* [Online]." Available: https://www.pluralsight.com/ courses/mining-data-text

[9] S. Rose, D. Engel, N. Cramer, and W. Cowley, "*Automatic Keyword Extraction from Individual Documents*" in *Text Mining: Applications and Theory*, 1st ed. Chichester, West Sussex, UK: John Wiley & Sons, 2010, ch. 1, pp.1 - 20, doi: 10.1002/9780470689646.ch1

[10] F. Pedregosa, et al. "*sklearn.feature_extraction.text.TfidfVectorizer*." scikit-learn. https://scikit-learn.org/stable/modules /generated/sklearn.feature_extraction.text.TfidfVectorizer.html. (accessed Nov. 29, 2020).

[11] M. Carpita, M. Sandri, A. Simonetto, P. Zuccolotto, "*Football Mining with R*", in *Data Mining Applications with R*. Brescia, Italy: Academic Press, 2014, ch. 14, pp. 397-433, doi: 10.1016/B978-0-12-411511-8.00015-3.

[12] F. Pedregosa, et al. "*sklearn.naive_bayes.MultinomialNB.*" scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html (accessed Nov. 29, 2020).

[13] K. Kirasich, T. Smith, B. Sadler, "*Random Forest vs Logistic Regression: Binary Classification for Heterogeneous Datasets*," SMU Data Science Review, vol. 1, no. 3, article 9, pp. 1-25, 2018, doi: , https://scholar.smu.edu/datasciencereview/vol1/iss3/9 (accessed Nov. 26, 2020)

[14] M. Stojiljković. "*Logistic Regression in Python*". RealPython. https://realpython.com/logistic-regression-python/ (accessed Nov. 26, 2020).

[15] A. Egg. "*RankNet: Learning to Rank from Pair-wise data*". Github. https://github.com/eggie5 /RankNet (accessed Nov. 26, 2020).

[16] K. Järvelin and J. Kekäläinen, "*IR evaluation methods for retrieving highly relevant documents*", in *Proc. 23rd ACM SIGIR Conf. R.& D. Info. Retrv.*,

ACM, Athens, Greece, Jan. 24-28, 2000, pp. 41-48,
doi: 10.1145/345508.345545.