Bibliometrics:
Predicting Publication Success

Team 4:
Tamanna Baig and **Jon Oakley**

Prepared for: Dr. Wang
CPSC 8650
April 15th, 2019

***Abstract*—Determining the impact of researchers has always fascinated scholars. Citation count is one of the most primitive bibliometrics that has been used to asses the success of a scientific publication. We presented an approach to predicting the success of an academic publication based on the features described in Table I. We reduced our initial corpus of 22,588 documents to 4,914 documents by removing any record that did not contain the required features. These documents were clustered into 27 clusters, and one cluster was omitted due to its size. Five of these clusters were chosen for analysis. the papers were marked as successful and unsuccessful based on whether or not they had more (successful) or less (unsuccessful) citations than the median number of citations for that cluster. The numeric features (Table I) were classified using a Random Forest classifier, and the text-based features (title, abstract, and keywords) were classified using three independent Multinomial Naive Bayes classifier. The predictions from these four classifiers were combined using an ensemble Random Forest classifier. Through this methodology, we were able to achieve an overall classification accuracy of 66.5% for the optimal clusters, and an overall classification accuracy of 57.7% for all of the clusters.**

***Index Terms*—Bibliometrics, Prediction, Citation Count, Success**

I. Introduction

Determining the impact of researchers has always fascinated scholars. Citation count is one of the most primitive bibliometrics that has been used to asses the success of a scientific publication. The h-index incorporates this idea and scores researchers based on the number of highly cited papers they have [1]. In [2] the h-index of a researcher was predicted using different attributes from their curriculum vitae (CV). Other predictors have used Twitter to predict the h-index change in authors with an 80% success rate [3]. However, it has been noted that there are many biases in the h-index, such as the age of the researcher [4].

An overview of recent bibliometric analyses is presented in [5]. A model for the long-term impact of a publication is presented by [6]. A study of accumulated advantage in academic endeavors was presented in [7]. By analyzing relationships of authors using a co-authorship network, [8] were able to predict future success of authors. Collaboration relationships were also found to be a predictor of success in [9]. Scholar trajectory was mapped in [10] in order to predict researchers migrating to different institutions. In [11] author characteristics were used to predict the success of academic publications.

Other studies used paper characteristics to predict their success. In [12], it was found that shorter titles were a positive predictor of success. Other research found that declarative titles didn't bias readers [13]. Author count and abstract complexity was also identified as a positive predictor [14]. Abstract complexity as a predictor was corroborated by [15]. Keywords were also identified as a predictor in [16]. Alternatively, full-text features were also useful in predicting the success of publications [17].

Figure 1 shows how the following sections fit together. Section II details the steps taken to clean and cluster the dataset. Section III discusses the visualizations used to inform our prediction methodology. Section IV discusses the various classification approaches used to predict the success of an academic publication, and Section V shows the results achieved with those classifier. Section VI provides closing thoughts and discusses future work. Code and supplemental results are provided in the Appendix.

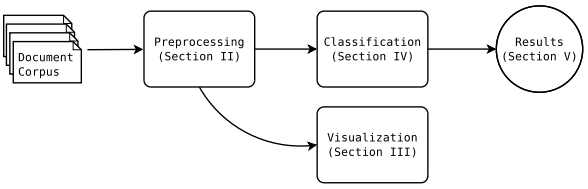

Fig. 1. Graphical abstract illustrating the tasks performed.

TABLE I
Features used for predicting the success of a publication.

Feature	Type	Meta?
title	str	✗
abstract	str	✗
keywords	str	✗
title_length	int	✓
abstract_length	int	✓
keyword_length	int	✓
year	int	✗
author_count	int	✓
page_count	int	✓
n_citation	int	✗

II. Preprocessing

The preprocessing stage involved cleaning the data and making it usable for our analyses. The original data set was sparse–many papers had missing fields, and there was no guarantee that a paper contained every field in the complete set of fields, shown in Table VIII in the Appendix. We selected the fields (and meta-fields) shown in Table I based on availability and the existing literature discussed in Section I.

Processing the original dataset was broken down into two sub-tasks: cleaning and clustering the dataset. The process used to clean and reduce the original dataset is discussed in Section II-A, and the process used to cluster relevant papers is described in Section II-B.

A. Cleaning

The cleaning processes is depicted in Figure 2. All existing features were extracted from the document corpus. The original document corpus was a Python dictionary, so Python was used to extract and process the papers. These features were analyzed for the relevance to success using the existing literature. The features shown in Table I were chosen, and all papers that did not contain those attributes were dropped. The initial corpus contained 22,588 papers, but only 4,914 papers contained all of the relevant features. Once the dataset contained only papers that had all the relevant

features, the meta-features were extracted. We define meta-features as features that depend on one or more features. For instance, `page_count` is considered a meta-feature since it depends on `page_start` and `page_end`. The inclusion of some meta-features reduced the number of documents in the final corpus more than regular features since they required several features to be present (otherwise the paper would be removed from the corpus). Other meta-features only depended on features already required (such as `title_length`). The final corpus was again stored in a Python dictionary for clustering. The script to perform this task is shown in Listing 1.

B. Clustering

In order to ensure our results were meaningful, we used k-means clustering to group similar papers for classification. There were two primary challenges: 1) determining the appropriate field to cluster based on, and 2) determining the appropriate value of K. Emperically, we found the `abstract` was the most effective field to use for clustering. In order to chose an appropriate value of K, we iterated over a number of possible values and chose the most appropriate based on the *inertia* (the sum-squared error in each of the centroids). The process for clustering the papers is shown in Figure 3. Python's `sklearn` machine learning library was used for all data processing in this work. The `TfidfVectorizer` was used to compute the TF-IDF value for each abstract, and the `KMeans` clustering function was used to find the clusters given the TF-IDF values.

After iterating over a number of values for K, we plotted the values, as shown in Figure 4. The optimal K was chosen because it occurred before the jitter occurred in the inertia values. Additionally, only one cluster needed to be excluded due to size. Cluster 0 contained only 15 papers and was therefore excluded.

With an appropriate K chosen, the papers were clustered according to the process depicted in Figure 3. The complete list of clusters is shown in the Appendix in Table IX. In this work, we consider the five clusters in Table II. Based on the keywords, we can see that these clusters are relatively compact. We also present the overall accuracy across all classes for comparison, but note that Cluster 0 was omitted due to insufficient sample size. The script to perform this task is shown in Listing 2.

III. Visualization

Visualization is the graphical representation of information and data. By using visual elements like graphs and charts, data visualization tools provide an accessible way to see and understand any trends, outliers or patterns present in the dataset.

For this project, various features like total number of the publications each year, authors with most publications, the papers to get the most citations and the institutions with the most affiliations are used to show any trends and patterns in the form of bar charts.

TABLE II
Clusters and a sample of their respective keywords.

Cluster	Count	Keywords
4	106	video, videos, 3D, quality, streams, users
15	170	query, XML, search, data, databases
17	869	design, people, user, information, research
20	93	internet, TCP, network, protocol, congestion
22	60	privacy, private, data, information, awareness

Figure 5 shows the total number of papers published each year. As depicted by the bars, it appears to be normally distributed, with 2010 seeing the highest publications. The number of publications by an author also play an important role in the success of an academic publication. Keeping this in mind the plot in Figure 6 depicts the Top 10 authors from the dataset with the highest number of publications. Another direct factor to predict the success is the citation count, and Figure 7 shows the highest number of citations received by a paper each year. Although the assumption would be that earlier the papers more the number of citations, but the plot tends to shed light that the year is not supposed to be the only factor in predicting so. Lastly, Figure 8 represents the top ten institutions/organizations in the world with highest number of affiliations, Carnegie Mellon University leading the line with around 874 affiliations.

IV. Classification

We break each cluster into two classes: unsuccessful and successful. We define a successful publication any publication that receives more than the median number of citations for that cluster. There are two important points to this: 1) the use of median as the average allows us to ensure the classes are balanced, and 2) we use the median relative to the cluster to control for some clusters having inherently more citations (due to popularity). The classification workflow is shown in Figure 9.

From Table I, we see that there are three string-based fields, and the rest are numeric. We use a Random Forest Classifier for the numeric fields, and we use a Multinomial Naive Bayes classifier for the text-based fields (title, abstract, and keywords). We use an independent classifier for each text-based field, so there is a title classifier, abstract classifier, and keyword classifier. In order to reach a consensus among the classifiers, we use another Random Forest classifier as an ensemble classifier. We give this classifier the prediction from all the other classifiers, and let it predict the resulting class. We train and optimize the classifiers for each cluster, so the parameters are not generalizable and are specific only to the designated cluster.

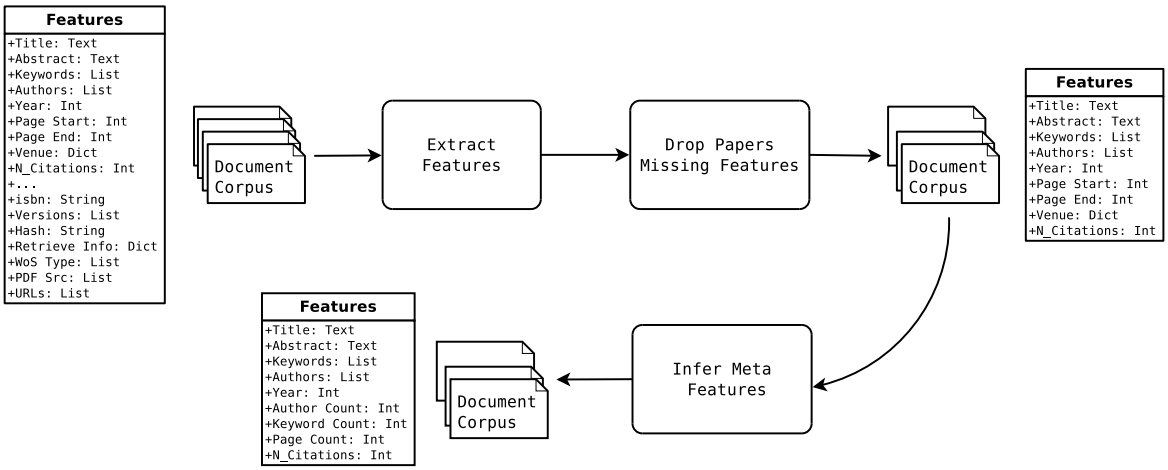

Fig. 2. Process used to clean the dataset.

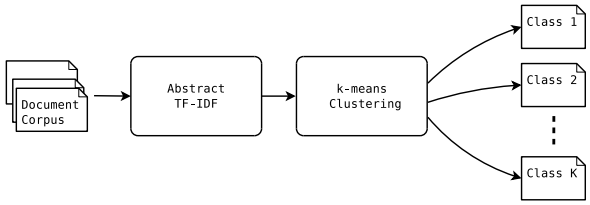

Fig. 3. Process used to cluster the papers.

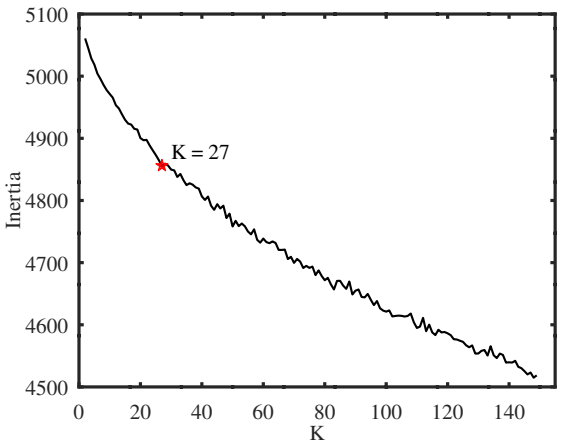

Fig. 4. Centroid inertia values for different K.

For evaluation purposes, we consider accuracy as the primary metric. We define accuracy as shown in Equation (1). This effectively measures the percent of correctly predicted values (true positive plus true negative divided by the sum of all occurrences).

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{1}$$

A. Training Method

Figure 10 shows our training methodology. For the text-based data, we consider the term frequency inverse document

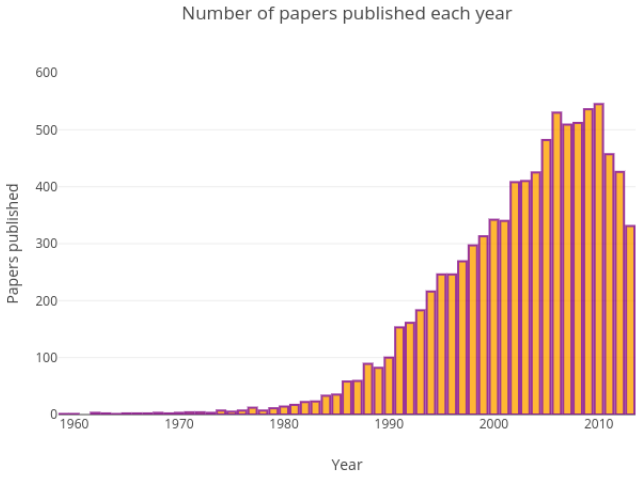

Fig. 5. Number of papers published each year.

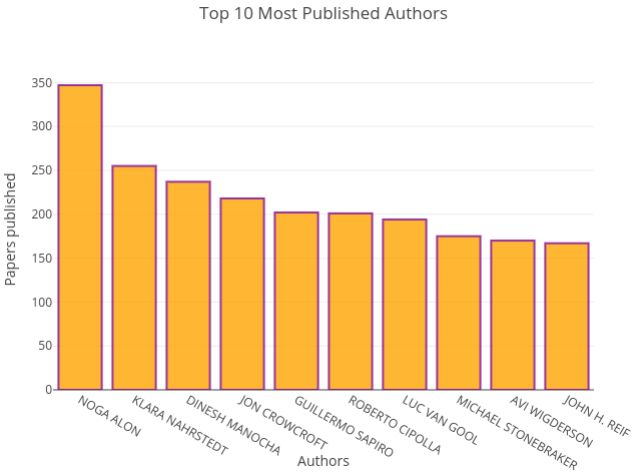

Fig. 6. Most prolific authors from the dataset.

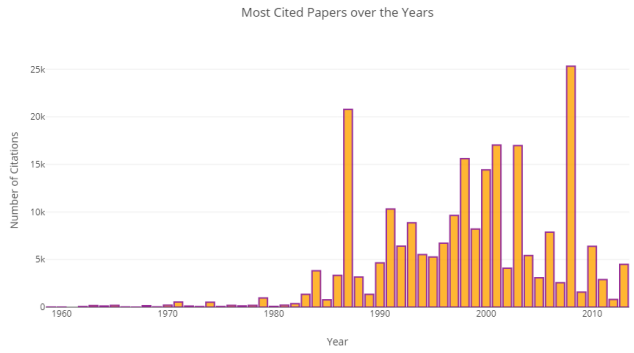

Fig. 7. Most citations of a paper published in a given year.

TABLE III
PARAMETERS FOR THE NUMERIC RANDOM FOREST CLASSIFIER.

	Numeric Classifier Parameters	
Cluster	n_estimators	max_features
4	10	2
15	10	6
17	8	2
20	12	2
22	6	4

frequency (TF-IDF) with stop-word removed. This is obtained using the `TfidfVectorizer` from `sklearn`. The numeric attributes are scaled using the `StandardScaler` from the `sklearn` library.

Before training the classifiers, the data from the clusters was formatted into a form that could be used by the classifier. The script to perform this task is shown in Listing 3. The data are split using the `test_train_split` function from `sklearn`, and the data are randomly split in half (50%). As shown in Figure 10, the training half of the data are used to train and optimize each classifier. The optimal parameters for the Random Forest classifier are found using the `GridSearchCV` function from `sklearn` and iterating over the following parameters: n_estimators and max_features. A five-fold cross validation is used with the grid search to ensure the parameters are in fact optimal. Once the classifiers are trained, the predicted values for the training data are used to train the ensemble classifier. As previously mentioned, this is also a Random Forest classifier, so the same `GridSearchCV` function is used to find the optimal parameters. The script to perform this task is shown in Listing 4.

V. RESULTS

The optimal parameters for the numeric Random Forest classifier are shown in Table III, and the optimal parameters for the ensemble Random Forest classifier are shown in Table IV.

With both classifiers, we can see there is some variation in the optimal parameters between clusters, emphasizing the importance of training the classifiers for each cluster independently. After training the Random Forest classifiers, we can see

TABLE IV
PARAMETERS FOR THE ENSEMBLE RANDOM FOREST CLASSIFIER.

	Ensemble Classifier Parameters	
Cluster	n_estimators	max_features
4	1	1
15	4	1
17	1	1
20	1	1
22	1	1

TABLE V
FEATURE WEIGHTS FOR THE NUMERIC RANDOM FOREST CLASSIFIER.

	Numeric Feature Weight					
Cluster	Title (length)	Abstract (length)	Keyword (length)	Year	Author Count	Page Count
4	0.089	0.117	0.143	0.240	0.188	0.223
15	0.160	0.157	0.121	0.321	0.065	0.175
17	0.136	0.253	0.174	0.183	0.089	0.165
20	0.087	0.214	0.121	0.242	0.109	0.228
22	0.140	0.038	0.066	0.556	0.155	0.045

how the feature importance weights the classification. Table V shows the weights for the numeric Random Forest classifier, and Table VI shows the weights for the ensemble Random Forest classifier.

Again, we can see the feature weights change between each cluster for both classifiers, emphasizing the importance of individual training. It is worth noting that year is the most important feature in Clusters 4 ,15,20, and 22, while abstract length is the most important feature in Cluster 17. It is also worth noting that in most cases, the numeric classifier is a sufficient predictor, but in some cases, there is noticeable improvement by including the other classifiers. The classification accuracy for each cluster are shown in Table VII.

The most important column in Table VII is the ensemble accuracy, as that is the effective accuracy for the cluster. As we can see, each of these clusters had an overall accuracy greater than 60%, but in all cases, there is room for improvement. As a final metric, we also consider the overall accuracy. We calculate this accuracy by summing the true positives, true negative, false positive, and false negatives from the ensemble classifiers for each cluster. This gives us an effective accuracy

TABLE VI
FEATURE WEIGHTS FOR THE ENSEMBLE RANDOM FOREST CLASSIFIER.

	Ensemble Feature Weight			
Cluster	Numeric Classifier	Title Classifier	Abstract Classifier	Keyword Classifier
4	1	0	0	0
15	0.420	0.271	0.059	0.250
17	0.917	0	0.083	0
20	1	0	0	0
22	1	0	0	0

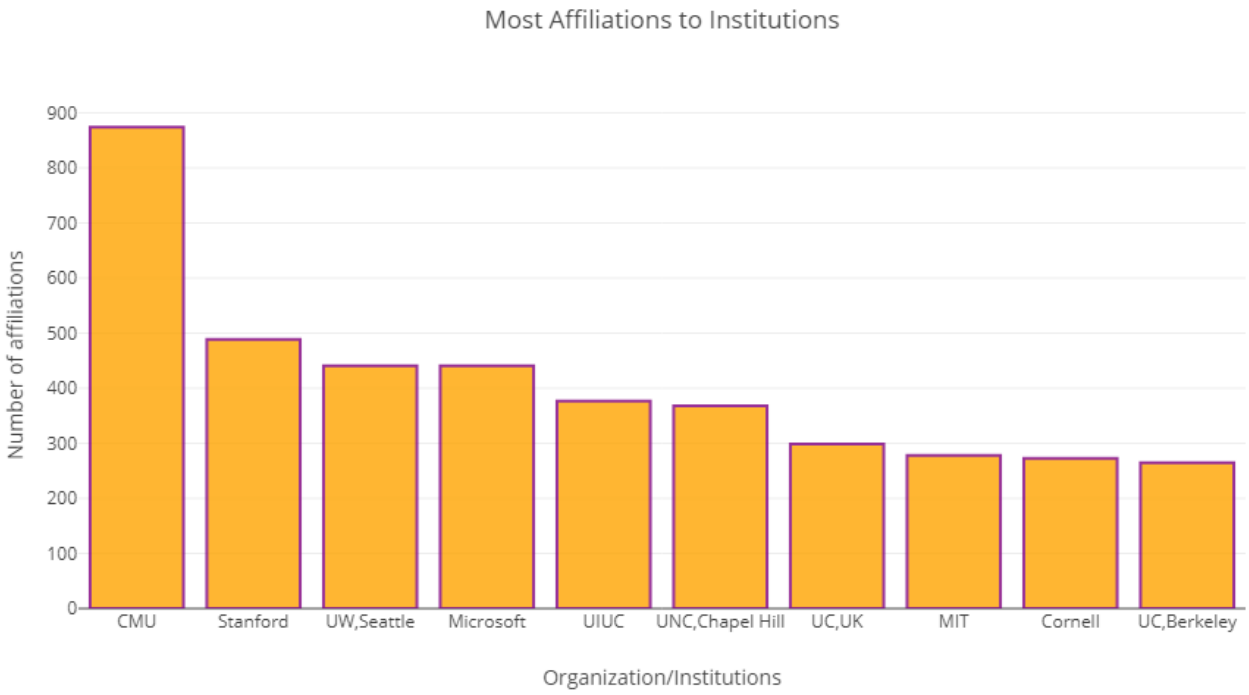

Fig. 8. Institutions with the most publications.

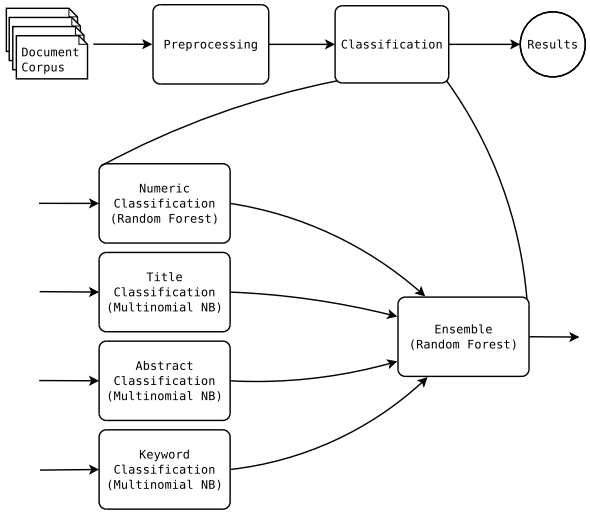

Fig. 9. Process for classifying papers in each cluster.

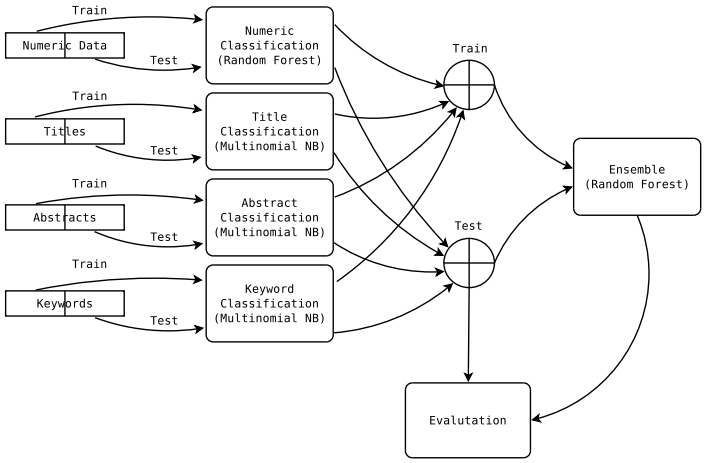

Fig. 10. Process for training the classifiers.

across the classes. This was useful for evaluating various classifiers, and providing an idea of *average* performance. The overall accuracy across the clusters in Table VII was 66.5%. For comparison, the overall accuracy across all clusters (except Cluster 0) was 57.5%. This provides an indication into the importance of the initial clustering.

TABLE VII
CLASSIFICATION ACCURACY FOR EACH CLUSTER.

	Accuracy (%)				
Cluster	Numeric	Title	Abstract	Keywords	Ensemble
4	60.4	45.8	56.3	58.3	60.4
15	63.0	64.2	65.4	55.6	61.7
17	64.2	59.1	59.6	62.5	68.5
20	64.3	42.9	35.7	38.1	64.3
22	64.3	0.5	0.5	0.5	64.3

VI. Conclusions

We presented an approach to predicting the success of an academic publication based on the features described in Table I. We reduced our initial corpus of 22,588 documents to 4,914 documents by removing any record that did not contain the required features. These documents were clustered into 27 clusters, and one cluster was omitted due to its size. Five of these clusters were chosen for analysis. the papers were marked as successful and unsuccessful based on whether or not they had more (successful) or less (unsuccessful) citations than the median number of citations for that cluster. The numeric features (Table I) were classified using a Random Forest classifier, and the text-based features (title, abstract, and keywords) were classified using three independent Multinomial Naive Bayes classifier. The predictions from these four classifiers were combined using an ensemble Random Forest classifier. Through this methodology, we were able to achieve an overall classification accuracy of 66.5% for the optimal clusters, and an overall classification accuracy of 57.7% for all of the clusters.

We expect this accuracy to drastically improve with a better dataset. We were limited to the available data set, which contained a wide array of papers with no particular focus. However, it is likely these results would be generalizable to a particular journal, given the paper fell within the same cluster. Future work will investigate alternative clustering algorithms and ways to incorporate title and keywords into the clustering. Future work will also consider additional numeric metrics, authors, and institutions.

Our accuracy for the optimal clusters is promising. While there is still room for improvement, we show our classification algorithm can accurately predict every two out of three successful publications, which offers realistic insight into the future success of an academic publication.

References

[1] J. E. Hirsch, "An index to quantify an individual's scientific research output," *Proceedings of the National academy of Sciences*, vol. 102, no. 46, pp. 16 569–16 572, 2005.

[2] D. E. Acuna, S. Allesina, and K. P. Kording, "Future impact: Predicting scientific success," *Nature*, vol. 489, no. 7415, p. 201, 2012.

[3] Z. Luo, J. Chen, and X. Liu, "Real-time scientific impact prediction in twitter," in *CCF Conference on Big Data*. Springer, 2018, pp. 108–123.

[4] O. Penner, R. K. Pan, A. M. Petersen, K. Kaski, and S. Fortunato, "On the predictability of future impact in science," *Scientific reports*, vol. 3, p. 3052, 2013.

[5] F. Radicchi and C. Castellano, "Understanding the scientific enterprise: citation analysis, data and modeling," in *Social Phenomena*. Springer, 2015, pp. 135–151.

[6] D. Wang, C. Song, and A.-L. Barabási, "Quantifying long-term scientific impact," *Science*, vol. 342, no. 6154, pp. 127–132, 2013.

[7] M. Perc, "The matthew effect in empirical data," *Journal of The Royal Society Interface*, vol. 11, no. 98, p. 20140378, 2014.

[8] E. Sarigöl, R. Pfitzner, I. Scholtes, A. Garas, and F. Schweitzer, "Predicting scientific success based on coauthorship networks," *EPJ Data Science*, vol. 3, no. 1, p. 9, 2014.

[9] T. Amjad, Y. Ding, J. Xu, C. Zhang, A. Daud, J. Tang, and M. Song, "Standing on the shoulders of giants," *Journal of Informetrics*, vol. 11, no. 1, pp. 307–323, 2017.

[10] Z. Shao, J. Tang, Y. Zhang, B. Gao, and Y. Wang, "Scholar trajectory: Visualizing the migration of talents," 2018.

[11] W. F. Laurance, D. C. Useche, S. G. Laurance, and C. J. Bradshaw, "Predicting publication success for biologists," *BioScience*, vol. 63, no. 10, pp. 817–823, 2013.

[12] A. Letchford, H. S. Moat, and T. Preis, "The advantage of short paper titles," *Royal Society open science*, vol. 2, no. 8, p. 150266, 2015.

[13] E. Wager, D. G. Altman, I. Simera, and T. P. Toma, "Do declarative titles affect readers' perceptions of research findings? a randomized trial," *Research integrity and peer review*, vol. 1, no. 1, p. 11, 2016.

[14] J. Sienkiewicz and E. G. Altmann, "Impact of lexical and sentiment factors on the popularity of scientific papers," *Royal Society open science*, vol. 3, no. 6, p. 160140, 2016.

[15] A. Letchford, T. Preis, and H. S. Moat, "The advantage of simple paper abstracts," *Journal of Informetrics*, vol. 10, no. 1, pp. 1–8, 2016.

[16] S. Uddin and A. Khan, "The impact of author-selected keywords on citation counts," *Journal of Informetrics*, vol. 10, no. 4, pp. 1166–1177, 2016.

[17] K. McKeown, H. Daume III, S. Chaturvedi, J. Paparrizos, K. Thadani, P. Barrio, O. Biran, S. Bothe, M. Collins, K. R. Fleischmann *et al.*, "Predicting the impact of scientific concepts using full-text features," *Journal of the Association for Information Science and Technology*, vol. 67, no. 11, pp. 2684–2696, 2016.

Appendix

TABLE VIII
ALL FIELDS PRESENT IN THE ORIGINAL DATASET.

Feature	Type
title	str
authors	list
n_citation	int
year	int
id	str
r	int
labels	list
volume	str
keywords	list
issue	str
abstract	str
versions	list
hash	str
lang	str
doi	str
venue	dict
page_start	str
isbn	str
pdf	str
issn	str
page_str	str
title_zh	str
pdf_src	list
hash_zh	str
page_end	str
keywords_zh	list
urls	list
abstract_zh	str
v_versions	list
retrieve_info	dict
wos_type	list
classes	list
n_wos_citation	int
i	str

TABLE IX
CLUSTERS AND A SELECTION OF THEIR MOST PREVALENT FEATURES.

Cluster	Features
0	alice, bob, girls, programming
1	storage, memory, data, server
2	translation, word, chinese, english
3	peer, p2p, peers, streaming
4	video, videos, 3d, quality
5	social, users, content, user
6	web, search, pages, page
7	data, algorithms, large, paper
8	learning, training, text, classification
9	algorithm, collision, present, compute
10	qos, service, multimedia, applications
11	traffic, network, flows, packet
12	motion, surface, 3d, algorithm
13	image, object, images, recognition
14	detection, malware, attacks, attack
15	query, queries, xml, search
16	graph, graphs, vertices, edges
17	design, people, user, information
18	retrieval, recommender, systems, information
19	log, time, algorithm, size
20	internet, tcp, network, protocol
21	problem, algorithm, algorithms, time
22	privacy, private, data, information
23	security, secure, protocols, protocol
24	devices, mobile, device, users
25	routing, network, networks, nodes
26	ontology, owl, semantic, logic

```python
#!/usr/bin/env python3
import pickle

ATTR_LIST = ['title','abstract','keywords','year','authors','page_start','page_end','venue','n_citation']
NAME_ATTR_LIST = ['name','aff']
VENUE_ATTR_LIST = ['name']

def process_paper(p):
    # Returns None if the paper is missing ANY of the attributes
    if not all(a in p.keys() for a in ATTR_LIST):
        return None
    if not all(p[k] for k in ATTR_LIST):
        return None
    # Affliliation attribute present and not empty
    if not all(k in author.keys() for author in p['authors'] for k in NAME_ATTR_LIST):
        return None
    if not all(author[k] for author in p['authors'] for k in NAME_ATTR_LIST):
        return None
    # Venue Attribute present and not empty
    if not all(a in p['venue'].keys() for a in VENUE_ATTR_LIST):
        return None
    if not all(p['venue'][k] for k in VENUE_ATTR_LIST):
        return None

    # Copies only the attributes in the list from the old paper to the new paper
    new_p = {k:v for (k,v) in p.items() if k in ATTR_LIST and v}
    if not all(a in new_p.keys() for a in ATTR_LIST):
        return None
    else:
        return new_p

def transform(p):
    # Authors
    authors = p['authors']
    author_list = [a['name'].upper() for a in authors]
    p['authors'] = author_list
    # Affiliations
    if not any(not 'aff' in a.keys() for a in authors):
        aff_list = [a['aff'].upper() for a in authors]
        p['aff'] = aff_list
    # Page count
    if p['page_start'].isnumeric() and p['page_end'].isnumeric():
        page_cnt = int(p['page_end']) - int(p['page_start'])
        del p['page_start']
        del p['page_end']
        p['page_count'] = page_cnt + 1
    else:
        p['page_count'] = None
    p['venue'] = p['venue']['name'].upper()

def hist_attr(papers,attr):
    vals = {}

    for p in papers.values():
        if not p[attr] in vals.keys():
            vals[p[attr]] = 1
        else:
            vals[p[attr]] += 1

    for k,v in sorted(vals.items(), key=lambda kv: kv[1],reverse=True):
        print("%s: %d" % (k,v))

if __name__ == "__main__":
    filename = 'data/trajectory_data.list'
    with open(filename, 'rb') as data_source:
        data = pickle.load(data_source)

```

```
    papers = {}
    # List of publications that meet the criteria
    for author in data:
        for p in author['pubs']:
            pID = p['id'].upper()
            # Create a deep copy with the specified attributes
            new_p = process_paper(p)
            if new_p and not pID in papers.keys():
                transform(new_p)
                papers[pID] = new_p

    print("Processed: %d papers" % len(papers))
    with open('data/database.pickle', 'wb') as f:
        pickle.dump(papers, f)
```

Listing 1. Python script to extract the papers from the original dataset and store papers containing all the required features in a new database.

```
#!/usr/bin/env python3
import pickle
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
import numpy as np

if __name__ == "__main__":
    filename = 'data/database.pickle'
    with open(filename, 'rb') as data_source:
        papers = pickle.load(data_source)

    dataset = [p['abstract'] for p in papers.values()]
    titles = [p['title'] for p in papers.values()]

    # Clustering!
    vectorizer = TfidfVectorizer(max_df=0.5, max_features=10000,
                                 min_df=2, stop_words='english',
                                 use_idf=True)
    X = vectorizer.fit_transform(dataset)

    print("n_samples: %d, n_features: %d" % X.shape)
    print()

    # Determined experimentally
    exp_k = 27

    km = KMeans(n_clusters=exp_k, init='k-means++', max_iter=100, n_init=1,
                verbose=True, random_state=5)

    print("Clustering sparse data with %s" % km)

    km.fit(X)
    print(km.random_state)
    groups = km.predict(X)

    order_centroids = km.cluster_centers_.argsort()[:, ::-1]
    terms = vectorizer.get_feature_names()

    with open('data/pp-fit.pickle', 'wb') as f:
        pickle.dump(groups, f)

    for g in range(min(groups),max(groups)+1):
        cluster = {}
        citations = []
        indices = np.argwhere(groups==g)

        # Print the relevant keywords and titles in the cluster
        print("Class: %d | Len: %d" % (g,indices.shape[0]))
        print("Terms:", end='')
        for t_idx in order_centroids[g,:10]:
            print(' %s' % terms[t_idx], end='')
        print()
```

```
        for idx in np.nditer(indices):
            print('\t' + titles[idx])

        # Add all the relevant papers to the cluster
        for idx in np.nditer(indices):
            k = list(papers.keys())[idx]
            if papers[k]['n_citation'] < 1000:
                cluster[k] = papers[k]
                citations.append(papers[k]['n_citation'])

        print("Median: " + str(np.median(citations)))

        with open('data/cluster-%d.pickle'%g, 'wb') as f:
            pickle.dump(cluster, f)
```

Listing 2. Python script to cluster the papers.

```
#!/usr/bin/env python3
import pickle
from sklearn.cluster import KMeans
import numpy as np
import sys

if __name__ == "__main__":
    if not len(sys.argv) == 2:
        print("Usage: pp-transform <cluster-max>")
        sys.exit()

    c_max = int(sys.argv[1])

    # Iterate over all clusters to process
    for c in range(0,c_max+1):
        print("Cluster: %d" % c)
        # Load the papers in that cluster
        filename = 'data/cluster-%d.pickle' % c
        with open(filename, 'rb') as f:
            papers = pickle.load(f)

        # Calculate the median citations in each cluster
        cite_list = [p['n_citation'] for p in papers.values()]
        median = np.median(cite_list)

        # Features: title_count, abstract_count, keyword_count, year, author_count, page_count
        X = np.empty([0, 6],dtype=int)
        # Features: Title, Abstract, Keywords
        X_text = []
        # Target
        Y = np.empty(0,dtype=int)
        for idx,p in enumerate(papers.values()):
            if not p['page_count'] == None:
                # Extracting the numerica data
                data = []
                data.append(len(p['title'].split(' ')))
                data.append(len(p['abstract'].split(' ')))
                data.append(len(p['keywords']))
                data.append(p['year'])
                data.append(len(p['authors']))
                data.append(p['page_count'])
                X = np.append(X,[data],axis=0)
                # Extracting the text data
                X_text.append([p['title'], p['abstract'], ' '.join(p['keywords'])])
                # Generating the ground truth
                if p['n_citation'] < median:
                    Y = np.append(Y,0)
                else:
                    Y = np.append(Y,1)

        with open('data/X-%d.pickle'%c, 'wb') as f:
```

```
            pickle.dump(X, f)
        with open('data/X-text-%d.pickle'%c, 'wb') as f:
            pickle.dump(X_text, f)
        with open('data/Y-%d.pickle'%c, 'wb') as f:
            pickle.dump(Y, f)
```

Listing 3. Python script to create a training set from the clusters.

```
#!/usr/bin/env python3
import pickle
import sys
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
# Import Classifiers
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import numpy as np

RAND_STATE = 5

if __name__ == "__main__":
    # Input clusters to process
    if not len(sys.argv) == 2:
        print("Usage: pp-transform <cluster-file>")
        sys.exit()

    class_txt = ''
    with open(sys.argv[1], 'r') as f:
        class_txt = f.read()

    total_tn = 0
    total_fp = 0
    total_fn = 0
    total_tp = 0
    # Iterate over clusters
    for c_str in class_txt.strip().split('\n'):
        # Load the cluster data
        c = int(c_str)
        print("Class %d" % c)
        with open('data/X-%d.pickle'%c, 'rb') as f:
            X = pickle.load(f)
        with open('data/X-text-%d.pickle'%c, 'rb') as f:
            X_text = pickle.load(f)
        with open('data/Y-%d.pickle'%c, 'rb') as f:
            Y = pickle.load(f)

        # Split numeric data
        X0_train, X0_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.5, \
                random_state=RAND_STATE)

        # Standard scale
        scaler = StandardScaler()
        scaler.fit(X0_train)  #only scale w/ respect to training data
        X0_train = scaler.transform(X0_train)
        X0_test = scaler.transform(X0_test)
        X = scaler.transform(X) #for validation on whole group

        # Title TF-IDF
        count_vec = CountVectorizer()
        X1_count = count_vec.fit_transform([x[0] for x in X_text])
        tfidf_transformer = TfidfTransformer()
        X1_tfidf = tfidf_transformer.fit_transform(X1_count)
        X1_text_train, X1_text_test, Y_train, Y_test = train_test_split(X1_tfidf, \
```

```
                Y, test_size=0.5, random_state=RAND_STATE)

        # Abstract TF-IDF
        count_vec = CountVectorizer()
        X2_count = count_vec.fit_transform([x[1] for x in X_text])
        tfidf_transformer = TfidfTransformer()
        X2_tfidf = tfidf_transformer.fit_transform(X2_count)
        X2_text_train, X2_text_test, Y_train, Y_test = train_test_split(X2_tfidf, \
                Y, test_size=0.5, random_state=RAND_STATE)

        # Keyword TF-IDF
        count_vec = CountVectorizer()
        X3_count = count_vec.fit_transform([x[2] for x in X_text])
        tfidf_transformer = TfidfTransformer()
        X3_tfidf = tfidf_transformer.fit_transform(X3_count)
        X3_text_train, X3_text_test, Y_train, Y_test = train_test_split(X3_tfidf, \
                Y, test_size=0.5, random_state=RAND_STATE)

        # Random Forest Search Parameters
        param_grid = [
                {'n_estimators': [1,2,4,6,8,10,12],'max_depth': [None],'
    min_samples_split': [2], 'max_features':[2,4,6], 'random_state': [0]}
        ]
        param_grid1 = [
                {'n_estimators': [1,2,4,6,8,10,12],'max_depth': [None],'
    min_samples_split': [2], 'max_features':[1,2,3,4], 'random_state': [0]}
        ]

        # Optimize numeric Random Forest classifier
        clf0 = GridSearchCV(RandomForestClassifier(), param_grid, cv=5, scoring='
    accuracy')
        clf0.fit(X0_train, Y_train)

        print("Detailed classification report:")
        print(clf0.best_params_)
        print()
        print("The model is trained on the full development set.")
        print("The scores are computed on the full evaluation set.")
        print()

        Y0_train_pred = clf0.predict(X0_train)
        Y_true, Y0_pred = Y_test, clf0.predict(X0_test)

        print(classification_report(Y_true, Y0_pred))
        print("ACCURACY: ", end='')
        print(accuracy_score(Y_true, Y0_pred))

        importances = clf0.best_estimator_.feature_importances_
        indices = np.argsort(importances)[::-1]
        for f in range(X.shape[1]):
            print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]])
    )

        # Title MNB Classifier
        clf1 = MultinomialNB().fit(X1_text_train, Y_train)
        Y1_train_pred = clf1.predict(X1_text_train)
        Y_true, Y1_pred = Y_test, clf1.predict(X1_text_test)
        print(classification_report(Y_true, Y1_pred))
        print("ACCURACY: ", end='')
        print(accuracy_score(Y_true, Y1_pred))

        # Abstract MNB Classifier
        clf2 = MultinomialNB().fit(X2_text_train, Y_train)
        Y2_train_pred = clf2.predict(X2_text_train)
        Y_true, Y2_pred = Y_test, clf2.predict(X2_text_test)
        print(classification_report(Y_true, Y2_pred))
        print("ACCURACY: ", end='')
        print(accuracy_score(Y_true, Y2_pred))

```

```
        # Keywords MNB Classifier
        clf3 = MultinomialNB().fit(X3_text_train, Y_train)
        Y3_train_pred = clf3.predict(X3_text_train)
        Y_true, Y3_pred = Y_test, clf3.predict(X3_text_test)
        print(classification_report(Y_true, Y3_pred))
        print("ACCURACY: ", end='')
        print(accuracy_score(Y_true, Y3_pred))

        # Ensemble Classifier
        clf4 = GridSearchCV(RandomForestClassifier(), param_grid1, cv=5, scoring='
    accuracy')
        Y4_train = np.vstack((Y0_train_pred,Y1_train_pred,Y2_train_pred,Y3_train_pred)
    ).T
        Y4_pred = np.vstack((Y0_pred,Y1_pred,Y2_pred,Y3_pred)).T
        clf4.fit(Y4_train, Y_train)

        print("Detailed classification report:")
        print(clf4.best_params_)
        print()
        print("The model is trained on the full development set.")
        print("The scores are computed on the full evaluation set.")
        print()

        Y_true, Y_pred = Y_test, clf4.predict(Y4_pred)

        print(classification_report(Y_true, Y_pred))
        print("ACCURACY: ", end='')
        print(accuracy_score(Y_true, Y_pred))

        importances = clf4.best_estimator_.feature_importances_
        indices = np.argsort(importances)[::-1]
        for f in range(Y4_pred.shape[1]):
            print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]])
    )

        # Accuracy calculation
        tn, fp, fn, tp = confusion_matrix(Y_true, Y_pred).ravel()
        total_tn += tn
        total_fp += fp
        total_fn += fn
        total_tp += tp

        print("OVERALL ACCURACY: ", end='')
        print((total_tp+total_tn)/(total_tp+total_fp+total_fn+total_tn))
```

Listing 4. Python script to classify the successful papers.