REP. OF TURKEY

TED UNIVERSITY

GRADUATE SCHOOL

INTERACTIVE COMPUTING AND INFORMATION SYSTEMS

IMPROVING DIVERSITY OF SEARCH RESULTS FOR THE NATIONAL LIBRARY OF TURKEY

CEYDA OKUYAN

ANKARA, 2021

IMPROVING DIVERSITY OF SEARCH RESULTS FOR THE NATIONAL LIBRARY OF TURKEY

A Thesis Submitted To
The Graduate School
of
TED University

by

CEYDA OKUYAN

In Partial Fulfillment of The Requirements
For
Master of Science
In
Interactive Computing and Informational Systems

ANKARA, 2021

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Ceyda Okuyan

Signature :

ABSTRACT

IMPROVING DIVERSITY OF SEARCH RESULTS FOR THE NATIONAL LIBRARY OF TURKEY

Ceyda Okuyan

M.S.; Interactive Computing and Informational Systems

Supervisor: Assist. Prof. Dr. Tayfun Küçükyılmaz

July, 2021

One of the main objectives of search systems is to improve the quality of the user experience by providing search results that are relevant to the user's information need. Providing a diverse set of search results contribute to the quality of service of a search system as it increases the possibility of satisfying the user's information need. Although literature suggests that diversification techniques are used for the Web search, they are not applied often to non-free text search systems. In this work, we propose a diversified search framework for library catalog search systems. The work presented here is tested on a real-world dataset acquired from the National Library of Turkey.

We have experimented the effects of changes in a wide range of parameters for our framework, testing the effectiveness, both in terms of search result diversity and result relevance. To this end, the proposed models adopt an automatic document categorization technique as a generic approach to the category discovery of the documents based on Latent Dirichlet Allocation (LDA). For diversification of the search results, we modified and used a Maximum Marginal Relevance (MMR)-based technique. The proposed framework achieved significant improvements on search result diversity (66.7%) while maintaining acceptable levels of relevance (1.9% loss).

Keywords: Diversification, Info-Gain, Maximum Marginal Relevance, MMR, Latent Dirichlet Allocation, LDA

ÖZ

MİLLÎ KÜTÜPHANE ARAMA SONUÇLARINDA KONU ÇEŞİTLENDİRME İYİLEŞTİRMESİ

Ceyda Okuyan

YÜKSEK LİSANS, İnteraktif Bilişim Sistemleri

Tez Yöneticisi: Dr. Öğr. Üyesi Tayfun Küçükyılmaz

Temmuz, 2021

Arama sistemlerinin temel amaçlarından birisi kullanıcıların bilgi ihtiyaçlarına yönelik arama sonuçları sağlayarak kullanıcı tecübesini iyileştirmektir. Çok çeşitli arama sonuçlarının sağlanması, kullanıcı ihtiyaçlarının yerine getirilme ihtimalini arttırarak, arama sistemi hizmet kalitesine katkı sağlar. Yaygın olarak web arama kullanılmasına rağmen, çeşitlendirme teknikleri serbest metin olmayan arama sistemlerine sıklıkla uygulanmazdı. Çalışmamızda, kütüphane katalog arama sistemleri için çeşitlendirilmiş arama altyapısı hazırladık. Hazırlanan çalışmayı, Milli Kütüphane'nin gerçek verileri ile test ettik.

Çalışmamızda, birkaç farklı yaklaşım hazırladık ve çalışma alt yapımızı hem sonuçların ilgililiği hem de sonuçların çeşitliliği açısından geniş bir parametre aralığında test ettik. Sonunda hazırlanan model dokümanların kategorilerini keşfetmek için genel bir yaklaşım olan LDA tabanlı bir otomatik doküman kategorizasyon tekniğini benimsedik. Arama sonuçlarını çeşitlendirmek için (Maksimum Uç İlgililik) MMR tabanlı teknik kullandık. Hazırladığımız altyapı arama sonuçlarının çeşitliliği üzerinde oldukça önemli ölçüde iyişileştirme (%66.7) sağlarken, ilgililiği (%1.9 kayıp) kabul edilebilir seviyede tutmuştur.

Anahtar Kelimeler: Çeşitlendirme, Info-Gain, MMR, LDA, Maksimum Uç İlgililik, Gizli Dirichlet Tahsisi

ACKNOWLEDGMENTS

I would like to express my gratitude to my thesis supervisor Assist. Prof. Dr. Tayfun Küçükyılmaz for his encouragement, support and belief in my work.

I would like to thank my thesis monitoring commitee and jury members Assoc. Prof. Dr. Tansel DÖKEROĞLU and Assoc. Prof. Dr. İsmail Sengör ALTINGÖVDE, for spending their time to read and comment on my thesis. I am grateful for their constructive comments.

I would like to thank former head of the National Library of Turkey, Zülfi TOMAN, for his support and permission to use the library data in my thesis. I also would like to thank the author of Kaşif system, Sedat SARIKAYA, for his technical support.

Finally, I would like thank my husband Erhan, father Erol, mother Hediye and brother Fatih Ahmet for their support and understanding throughout my thesis study. Last but not least, I want to thank my little daughters Nihal and İlayda whose mere existance helped me a lot to finish my thesis study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ABBREVIATIONS

MMR	Maximal Marginal Relevance
MSD	Max-Sum Dispersion
IR	Information Retrieval
LDA	Latent Dirichlet Allocation
TREC	Text Retrieval Conference

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Libraries provide an essential service to users, who desire to satisfy their information needs. Searching for materials in the libraries collections according to users' queries is the backbone of this service. Libraries have a large number of documents, written and stored in a variety of ways in their collections. Without good search functionality, users cannot find the documents they desire. A good search algorithm should return a list of relevant documents to users, so they can initiate their information need physically.

Relevance of search results is quite important. However, that is not the single factor towards user satisfaction. Users also want the outcome to cover a variety of results within related topics as well. In other words, good search results have to be both relevant and diverse. This way, users can examine a wide range of topics.

Diversification is a well-studied subject in the context of web search. Most search engines diversify their results. However, we observed that the library search algorithms for library search, like the ones that are used and adopted in the National Library of Turkey, only considered relevance. Thus, we wanted to work on diversified search algorithms for library dataset, as the diversification is a subject dependent topic. As library document collections contain physical entities, the diversification techniques for web search might not transfer well into library search. Thus, our work on diversified search algorithm for libraries will provide insight through improving the user experience in a library setting.

1.1. BACKGROUND

Knowledge production has started as early as thinking. At the time, the knowledge has accumulated in stacks. As the number of people who wanted to benefit from the resulting knowledge increased, these stacks were transformed into more accessible forms such as books, magazines, CD-ROMs etc. Over time, classification, cataloguing and serving the information becomes a necessity. Thus the libraries have been established.

Libraries are one of the earliest forms of information storage methods. They allowed people to keep and build upon the available knowledge, as well as take a function for transferring it to the next generations. Libraries are organizations that have existed since the 17th century BC. Modern, scientific and research libraries, have been around since 18th century. The first scientific and research library; Göttingen University Library, was founded in Germany in 1735.

Although the libraries still exist similar to their ancient forms today, they also evolved significantly. They are no longer a mere storage for documents, but also become an information management system. Currently, most libraries are computerized to some degree. While some use computer systems just to manage their inventory, some digitized all their materials and started serving completely online. Most libraries collect and catalogue the attributes of each document in their collections and use them for providing search facilities. (Keskinkilic, 2011)

There are several functionalities libraries provide. The processes that make up this system, called library operations, are as follows;

- Acquisition
- Classification
- Cataloging
- Binding (bookbinding)
- Racking up
- Availing of materials
- Literature Reviewing
- Lending/ Borrowing (Circulation)
- Periodicals
- Periodicals Acquisition
- Periodicals Following/Tracing
- Availing of Periodicals

Most of these functionalities are self-descriptive and out of the scope of this study. In this work, we only will concentrate on cataloging services as they relate to the context and categorization of a document in a library. Materials which are acquired by the libraries via various methods must be organized and registered according to certain rules. These procedures are considered as the main functions of cataloging. Cataloging is the process of preparing a bibliographic record and giving an identification number for the material. The purpose of cataloging is to inform users about the availability and the location of each material in the physical space.

Descriptive cataloging gives description of materials. It describes the Basic Entry of the library item; author's name, title of materials, year of publication, location of publication, publisher, subject headings, physical description etc. and then adds Additional Entry: second author and other authors, parallel material title (2nd title), URL entry, sequence/series recording. The user catalog, which is the product of the cataloging process, is a library's most important resource for relating to, locating and tracking the availability of resources. (Gültekin, Kılıç, Balta, Özdemir, & Karaağaç, 2016)

The advent of digital technology also affected the libraries. The information produced in today’s world are mostly digital. Accordingly, libraries also assumed the role of storing and serving such digital information. Information sources come to libraries in different formats. These diverse sources of information are processed with certain methods, depending on the information source. This digital transformation brings lots of topics such as digital libraries, information security, information retrieval, data storage etc. into the library context.

1.2. PROBLEM DEFINITION

Library query searches scan the collections and returns the documents that match with the user query as best as possible. This approach exploits the similarity between documents and the user query as the only objective. The variation within the result set is not considered as a secondary objective. Diversification aims to improve such variation. Some diversification techniques aim to select results that are as dissimilar to each other, while still being relatively similar to the query. Some techniques use categorization of documents. In such techniques, the search algorithm aims to select results from multiple categories while still selecting relevant documents. Diversification as an objective of search has a conflicting goal with relevance: Diversified search algorithms aim to maximize the user experience by providing as much diversity as possible within the result set while losing as little relevance as possible.

Figure 1.1: Example diverse book result set for query 'Kedi'

In the National Library of Turkey, when a user wants to search about a subject, user must use OPAC (online public access catalog) which is library search engine. For example, let's assume that a user types a query to retrieve a book about 'kedi'; the Turkish word for cat. Ideally, the final search result set of 'kedi' should contain books from different aspects, and within different categories such as children stories, cats-care, history, thesis of veterinarian science, graphic novel, hypnotherapy etc. A possible set of results is illustrated in Figure 1.1.

But many times, the final search result set include books exclusively from a single category. For example, Figure 1.2 illustrates an example result set of the query ‘kedi’, containing books only from the category children stories.

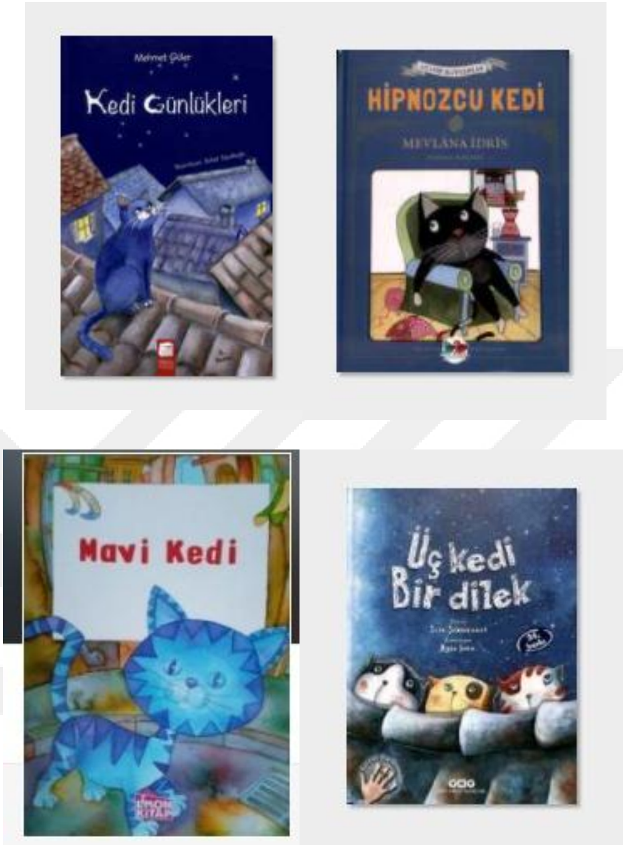

Figure 1.2: Example non-diverse book result set for query 'Kedi'

This example illustrates the importance of diversification clearly. Users are far more likely to find what they are interested from the diverse result set as opposed to a non-diverse set of results. Also, user queries are not always professionally built and precise. Users tend to narrow the materials with a simple query and then search through the results for more specific needs they have. The diversified results are far better suited for this approach.

Diversification techniques are applied in several domains. However, for the library search domain, more specifically for National Library of Turkey’s search system, diversification was not used. Diversification is a domain specific technique. Its performance varies with the data it uses. Accordingly, diversification approaches that work well for web searches might not work well for other domains. The aim of this work is to develop a diversified search framework for the library searches.

1.3. CONTRIBUTIONS AND NOVELTIES

Diversification is a domain dependent technique. Many approaches and hyper-parameters must be fine-tuned for the domain they are developed for. The main contribution of this work is to develop a diversified search framework for library data. For this, we need to develop a search algorithm that takes diversification into consideration. The search algorithm we develop is not a novel algorithm. We based our search algorithm on a few existing algorithms.

Our main contribution is to develop a diversified search framework for library data. Then we need to fine-tune the hyper-parameters using real library queries. This way the resulting search algorithm would perform well. Our last contribution is to analyze and show how does the diversified search algorithm perform compared to the currently used search system in the National Library of Turkey and evaluating the effects of parameters governing the diversified search algorithm.

1.4. DATASET

The dataset used in this work is the catalog database of the National Library of Turkey. The real user queries made on the National Library's search interface are also recorded and used in this work.

As mentioned earlier, it is compulsory by law for any printed material produced within Turkey, to be submitted to assembly libraries. National Library of Turkey is one of these assembly libraries. Any printed material arriving to the National Library of Turkey is processed and presented to users. One of the most important steps while processing the material is the cataloging.

During the cataloging, the detailed information about the material is entered into a database following very specific rules. The title of the material, the author, publisher and date are some examples input that is recorded during cataloging. Some information that is recorded is material-specific. The hand written materials have much different attributes than periodically printed materials and each material type is cataloged appropriately. The search engine used in the National Library of Turkey works on this catalogue database and answers the queries of libraries on-site and off-site users.

In this thesis, we used the exported catalogue information of materials that arrived at the National Library of Turkey initially and collected until year 2018. Even materials from the Ottoman era can be found in the dataset. We also acquired the user queries logged by the system until the year 2018. These user queries are logged by the "Kaşif" online library search system which became active in year 2016 (Sarikaya). Thus, query dataset covers only two years of queries. However, these queries are relatively recent and numerous enough for our purposes.

Although the catalogue database contains much information about the materials, we only used a subset of the descriptions that are allowed to be used and deemed useful for research purposes: We did not request information that will not be useful in our work, such as the physical dimensions of the material. The information on materials that are collected and used during this study include: unique shelf number, Title, Author, publish year and the Dewey code of the material. The dataset used in the work is obtained with the written permission of former Head of National Library of Turkey; Zülfi Toman, for the purpose of research use.

In the dataset, there are roughly 1.15 million documents. Note that the terms material and document relate to an entity for which result relevance is calculated, and is used interchangeably throughout this study. We also collected roughly 166 thousand user queries. Document dataset is in excel format with 289Mbytes of size, while query dataset is in excel format and of 11 Mbytes of size.

The collection of the National Library of Turkey is quite diverse and the users are also coming from diverse backgrounds. Accordingly, this dataset represents a real world cataloging service. Table 1.1 present a small sample of materials from the catalogue dataset and Table 1.2 present a sample of user queries from the queries dataset.

Table 1. 1: Sample materials from the library catalogue dataset.

Unique Shelf Number	Title	Author	Dewey Code	Publish Year
2012 AD 33561	İmparatorluk /	Hardt, Michael,	320.51, 325.32	2012
2012 BD 12405	DUS sorubankası :	Canpolat, Mehmet	617.6076	2012
2012 BD 12403	1001 hayvanıbulun /	Brocklehurst, Ruth	649.51	2012
2012 AD 33862	Yeniarkadaş /	Tücan, Erdoğan	808.06831	2012
2012 BD 12352	Miniklerokulabaşlıyor 1 /	Ceylan, Reyhan	649.51	2012
2014 AD 10953	Mavidünyam :	Yanık, Mustafa	811.42	2012
2014 AD 13118	Gramsci çağı :	Thomas, Peter D	335.4, 320.5322092	2013
2013 AD 21114	Muhsine /	Altıntaş, Münevver,	813.42	2013
2012 AD 33852	Yeraltından notlar :	Dostoyevski, Fyodor Mihayloviç,	891.733	2012
2012 AD 33670	Beyazdiş /	London, Jack,	808.0683283	2012

Table 1. 2: Sample queries from the library user query dataset.

Criterion 1	Criterion 2
Konya yemekleri	
	Tur, İsmail Hakkı
Gözlerimin söylettiği : şiirler	
9756527153	
Gözlerimin söylettiği	
TARİHİ TÜRK LEHÇELERİNDE İSİMDEN İSİM YAPMA EKLERİ	SALİM KÜÇÜK
928.94335	
	Ertaş, Şeref
Medeniyet taihi	
Medeniyet tarihi	

The rest of this thesis is organized as follows:

- Chapter 2 reviews previous related works on information retrieval field and more specifically in the search result diversification problem.
- Chapter 3 details the proposed method. The framework, the preprocessing steps, the diversified search algorithm and the analysis algorithms are described in detail.

- Chapter 4 analyzes the experimental results, fine tunes the hyper-parameters and evaluates the overall success of the proposed method.
- Chapter 5 gives conclusion and discusses possible future work on this subject.

CHAPTER 2

RELATED WORK

In recent years, the amount of online information has grown significantly. Particularly for processing and handling the text data, text classification techniques become quite important. Information retrieval can be described as finding documents from a large collection of unstructured documents that satisfy some information based needs (Kowsari et al., 2019). Information Retrieval models try to retrieve as many related models as well as retrieving as few non-related models as possible. The information retrieval research describe how the documents are stored and represented, what kind of algorithms are used for ranking the results, used preprocessing steps and etc.

With the increase in usage of computers and World Wide Web, the printed catalogs of earlier times had to be converted into a digital format. However, classical database processing tools and methods was still used instead of information retrieval models: Searches were done using the information about the document itself, not using the content of the documents. With the advent in digital technologies, especially in terms of computational power, data storage techniques and the explosion in the amount of data on the web, the information retrieval field becomes increasingly important. Nowadays, Information retrieval systems can store trillions of documents including various formats like images, audio and video as well as text and still be able to execute many queries and retrieve related documents under several seconds. In general, users can compose queries in terms of free text, describing their information need. The words, or terms, in the query are used to classify every document with respect to whether a document being relevant to that word. Thus, a query consisting of multiple words combine the relevance scores of each individual word to each document. The information retrieval system then returns the list of documents, whose combined relevance score to the given query is highest (Clipa & Di Nunzio, 2020).

2.1. DIVERSIFICATION

The importance of search engines has increased with the advent of big data and World Wide Web. They are used for serving billions of people for providing their data needs each day. Search engines are software frameworks that provide a list of documents according to user query. They play a critical role in information retrieval activities (Göynük, 2019).

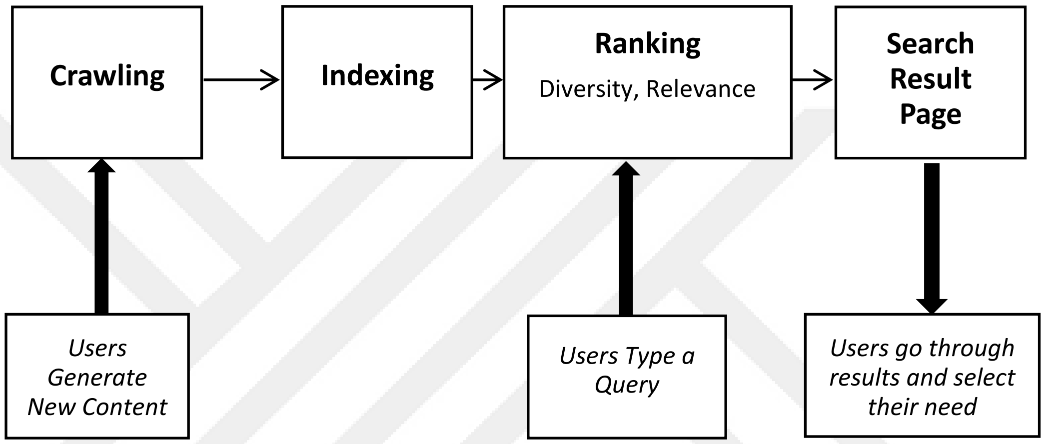

Figure 2.1: Main components of search engines

Search engines have three main tasks. The first one is crawling. Crawling is the discovery process of new and updated content. It is very important for expanding information content of the search engine. Crawling is done by robots called spider / web crawler. Web crawlers scan all websites on the internet. They collect the URLs of the content and move the information it collects to the servers of the search engine. The second job of search engines is Indexing. Indexing is basically processing of the data obtained by crawling. The data is categorized according to the information content and indexed so the access to information can be efficient. Without these indices finding desired information would be practically impossible for large datasets. The last job of a Search Engine is ranking. Ranking is the process of finding related documents to return upon a user query. With the ranking process, documents related to the user searches are ordered by relevance. He et al. describes a novel method for this ranking process based on query-specific cluster ranking (He, Meij, & Rijke, 2011). The results are usually shown to users through a user interface

(Göynük, 2019), (Leuken, Garcia, Olivares, & Zwol, 2009). The search result diversification is an additional consideration of ranking process. Wu et al. states that search result diversification is generally carried out in two steps (Wu, Zhang, & Xu,, 2019). At the first step, only the relevance of the documents according to a specific query is considered. In the second step, diversification algorithm re-ranks the results to make the results cover as much different topics as possible. However, this is a tradeoff between relevance and diversity. With the increase in data sizes, efficient retrieval of search results becomes increasingly important. Naini et al. describes a scalable and efficient method for search result diversification for web searches (Naini, Altingovde, & Siberski, 2016).

Search result diversification is an important problem. It aims to cover multiple aspects of a user query. Another use of diversification is to better handle ambiguous or multifaceted queries. With diversification multiple aspects of queries are covered, so the users can find something about the aspect they are interested in. Vieira et al. discusses about various techniques used in query result diversification (Vieira, et al., 2011). Most diversification techniques try to stabilize both the similarity and diversity in order to provide result diversification. There are two main approximate approaches for this problem; implicit result diversification and explicit result diversification (Ulu, 2019).

2.1.1. Implicit Diversification

Implicit diversification classifies documents using only the documents' implicit properties. They do not use any external information like details of the queries or some other external knowledge (Keswani & Celis, 2020). These techniques are the first techniques proposed in diversification context.

At the beginning of implicit diversification process, each document is represented as a feature vector. The features can be certain words, combinations of words or other properties of the documents. But with implicit diversification, features can only be generated from the implicit properties of documents. Features cannot be dependent on query properties. Implicit diversification tries to estimate how close the documents are by comparing the feature vectors representing the documents (Qaiser & Ali, 2018).

The one of the widely used example of implicit diversification is Maximal Marginal Relevance (MMR) algorithm, by Carbonell and Goldstein (Carbonelland & Goldstein, 1998).

$$MMR(q, R) = \alpha * \frac{\sum_{i=1}^{R.len} similarity(\mathrm{R[i],q})}{R.len} + (1 - \alpha) * diversity(R) \quad (1)$$

MMR algorithm tries to maximize both the average similarity and diversity. It simply tries to find a subset of documents among the corpus, such that both similarity and diversity is maximized. Since similarity and diversity are different objectives, they do not usually agree. Thus MMR uses a tunable parameter that combines these two objectives into one and tries to maximize it. MMR have many variations. Although in principle, they are all similar, they differ in details. We do not get into details of those variations, but explain one representative example.

MMR scans all the documents in the corpus and tries to find a subset of those documents whose objective function is maximized. The objective function is given in Eq. 1. It simply quantifies the value of a selected subset R, for a query q from the corpus. First part of the equation simply finds the average similarity between the query and each of the documents in the selected subset. Similarity between a document and a query is independent to the selected subset. Second part of the equation handles diversity. It simply represents how diverse the selected subset R is, by quantifying how many different classes are represented by selected documents. Average similarity and diversity valuations are combined by a linear combination constant α. Weights of similarity and diversity can be tuned via this parameter. How to find the document subset that will maximize the MMR objective function depends on implementation and the nuances of document representations and objective functions.

Another widely used implicit diversification result is Max-Sum Dispersion (MSD) (Gollapudi & Sharma, 2009). The MSD function given in Eq.2, works on a pair of documents. It simply tries to find a pair of documents that are as similar to given query as possible, while being as dissimilar to each other as possible. After finding the pair of such documents, they are added to the result set. Then the approach can be extended to add new documents to the result set continuing the same principle. We simply need to select the next document that is as similar to query, while being as dissimilar to already selected documents. Since documents that are as dissimilar to each other are selected as a result set, the diversity is achieved. The weighting between similarity to the query and dissimilarity to other documents in the result set can be linearly tuned via a tunable constant. This algorithm is a greedy algorithm. However, it is appealing due to its relatively lower computational overhead.

$$MSD(di, dj, q) = (1 - \alpha) * \left(sim(di, q) + sim(dj, q)\right) + 2\alpha * \left(1 - sim(di, dj)\right) \quad (2)$$

2.1.2. Explicit Diversification

Explicit diversification techniques benefit from external information such as query aspects, explicitly created by using information taxonomy or query logs (Agrawal, Gollapudi, Halverson, & Ieong, 2009)(Santos, Peng, Macdonald, & Qunis, 2010). Explicit diversification techniques handle diversification in a query oriented way. Initially, they try to identify every aspect of a given query, including many related subtopics and possible ambiguities etc. After that, these techniques try to cover all the aspects of the query by selecting documents of those aspects. Ozdemiray and Altingovde states that the primary difficulty of explicit methods is to find every aspects of the query to cover every possible information need (Özdemiray & Altıngövde, 2013). Yigit-Sert et al. present a method for explicit diversification of search results across multiple dimensions (Yigit-Sert, Altingovde, Macdonald, Ounis, & Ulusoy, 2021).

Intent-Aware Select is one of the well-known and earliest explicit diversification techniques. Agrawal et. al. proposed a method that is able to get relationships of queries and documents via the categories on taxonomy (Agrawal, Gollapudi, Halverson, & Ieong, 2009). With this classification, documents and queries can be represented and compared in category domain.

Explicit Query Aspect Diversification is another explicit diversification technique. It is based on probability of relevance and diversity (Yang, et al., 2017). It uses query aspects acquired by subtopics (Text Retrieval Conference (TREC) subtopics were used in the mentioned study).

2.2. FEATURE SELECTION

Features are individual quantifiable characteristics of documents. Classification of documents can be achieved by many machine learning algorithms. Feature selection is an important part of information retrieval process. Feature selection leads to a more generalized representation of text; via discovering relatively important variables, removing stopwords such as words that can appear anywhere such as “and”, and identifying correlating variables. This process helps to eliminate some of the variables and helps to reduce classification problem dimensionality. This way, the predictor performance can be improved and computation time can be reduced. In this thesis, we examined some feature selection methods found in the literature and analyzed their impact on the overall predictive performance of the algorithm. Main goal of the feature selection is to select a subset of features from the documents which can efficiently describe the input data while reducing the noise in the data and still be able to give good predictive performance. Filter based feature selection methods are used as preprocessing where the features are ranked and the high ranking ones are selected. Wrapper-based feature selection methods are based on predictor performance. A search algorithm is used to determine the best selection criterion that will give the best predictor performance. Embedded feature selection methods include the feature selection criterion into the training process (Chandrashekar & Şahin, 2014).

There are many datasets with various sizes. Some datasets are extremely large. Processing a very large dataset can be very challenging. It is common for text data and bag of words models that, many features are so scarcely used they only represent a few documents. Consequently, such features usually do not contain much predictive value. On the other hand, some features are so common they appear in a significant portion of the documents. Such features again, do not give any predictive value as they do not provide discriminatory information. Feature selection can be quite useful in such situations. A significant portion of features can be eliminated without inversely affecting the predictive performance. In fact, many studies indicate that feature selection, by improving the generalization of the data, may improve predictive performance (Manning, Raghavan, & Schütze, 2008).

As a summary, feature selection reduces the dimensionality of the documents and reduces the computation times significantly. It also acts like a noise filter for dataset improving the predictive performance. Accordingly, feature selection enables machine learning models to train faster. Since it acts as noise removal, it improves the accuracy and reduces overfitting. Thus feature selection is a very useful tool for information retrieval tasks (Visalakshi & Radha, 2014).

The goal of feature selection process is to find a set of features that describe the dataset as best as possible while containing least number of features. This selection is basically done by comparing how descriptive certain features are for distinguishing documents belonging to different classes. If a feature is useful for that purpose, then it is selected. Otherwise it is eliminated (Bagirov, Rubinov, & Yearwood, 2002). There are two major approaches for this problem; namely feature selection and feature extraction (Jiangand & Lee, 2007).

Feature selection methods work by selecting a subset of original features that will be used by classifiers. Features that are not selected are eliminated. A widely used example of this approach is the Information Gain (Yang & Pedersen, 1997) algorithm. It simply computes the cumulative predictive value of each feature and selection can be done according to these values.

Feature extraction methods synthesize new features from a combination of old features. They simply convert the old dataset to a new dataset with whole new feature sets and reduce the dimensionality while doing so. Word clustering techniques (Law, Figueiredoand, & Jain, 2004), are one of the widely used examples of this approach. The idea behind word clustering is to group the similar words and use a generic name instead of individual names. For example, instead of keeping the words “apple”, “orange” and “banana” in the dataset, we can group them and replace with the word “fruit”. If performed appropriately, this method can reduce the number of features drastically without affecting the predictive performance much (Jiangand & Lee, 2007).

Document collections are usually considered as containing unstructured and unlabeled data. It is possible to use unsupervised learning in order to extract the hidden structure within this data. Clustering algorithms (Law, Figueiredoand, & Jain, 2004) are the most common unsupervised learning techniques. They can discover natural groupings of certain features. Thus clustering based feature selection can be quite useful (Chandrashekar & Şahin, 2014).

2.2.1. Filtering Methods

Filtering methods relies on the distinctiveness of features within the dataset. These methods use distance, information, dependency, and consistency based assessment criterions in order to identify the usability of a feature. The filtering methods are ranking based methods. They are usually used as a preprocessing operation in order to find non-discriminating or dependant attributes and such variables that are found to be relatively less important are filtered out. They are mainly used for their simplicity and potential to produce relevant features. Chi-squared (Pearson, 1900) and information gain (Uğuz, 2011) are some commonly used filtering methods (Visalakshi & Radha, 2014).

Ranking methods use a scoring mechanism to quantify the relevance value of variables. Then a thresholding mechanism can be used to filter out less relevant features. Ranking methods can be considered as filter methods since they are applied before the classification. Scoring mechanism should quantify the relevance of each feature for distinguishing different classes (Chandrashekar & Şahin, 2014).

Chi Square: The Chi-Square method evaluates each feature by computing their chi square statistics with respect to classes. The chi square score represents the dependency between the feature and the class. The features with higher chi-square scores are more informative (Zarepoor & Seeja, 2015). The formula for Chi Square is given in (3). Chi Square value is represented as x^2 while O_i represent the observed value and E_i represent the expected value.

$$x^2 = \sum \frac{(O_i - E_i)^2}{E_i} \tag{3}$$

Information Gain: Information gain is a feature selection method that computes the value of each feature. The value of a feature represents the value of that feature for discriminating between classes. Then the features are simply selected or eliminated according to their scores (Zarepoor & Seeja, 2015).

Entropy of a dataset represents how predictable the dataset is. High entropy means the dataset is not predictable. Low entropy means dataset is quite predictable. Entropy for a dataset with C classes is defined as;

$$Entropy = -\sum_{i}^{C} p_i log_2 p_i \tag{4}$$

The information gain of a certain subset of features is defined as the difference of entropy in we remove the said subset. Formally;

$$IG(V, A) = Entropy(V) - Entropy(V - A) \tag{5}$$

The information gained by removing the features A, from the dataset V is defined as the differences of entropies of V and V-A.

2.2.2. Wrapper Methods

Wrapper methods require a machine learning algorithm to use its performance as evaluation criteria. Wrapper methods search for a set of features that will maximize the mining performance. Predictive accuracies of classification tasks are used for evaluation. Forward selection (Wah, Ibrahim, Hamid, Rahman, & Fong, 2018), backward elimination (Ladha & Deepa, 2011) and bi-directional elimination (Stepwise Selection) (R., 2012) are some examples of wrapper methods. (Luhaniwal, 2020)

2.2.3. Embedded Methods

Embedded methods handle the model training iteratively. At each iteration, they extract the features that are found to be the most important. Regularization methods are the most commonly used embedded methods. They penalize each feature to incentivize the model to select as few feature as possible. LASSO Regression (Rohini, 2016), Ridge Regression (McDonald, 2009) (Saurabh Paul, 2015) and Elastic Net (Zou, 2005) are some examples. (Brownlee, 2020)

Difference between filtering and wrapper methods: The difference between filter and wrapper methods can be confusing. There are a few points that differentiate those two. First, filter methods do not employ a machine learning model to distinguish useful and redundant features. Wrapper methods rely on machine learning algorithms for that purpose. Second, since wrapper methods require a machine learning model and some training they are computationally expensive. On the other hand, filter methods are much faster. Filter methods are based on statistical correlations. Thus they require quite large dataset size to be able to reveal minor relations between features statistically. Wrapper methods are based on machine learning algorithms and they also require large datasets. However, they can work with much smaller datasets and still be able to provide the best subset of features, at least for the given limited dataset, where filter methods might fail to do so. The power of wrapper methods to find the best feature set has a downside. They might lead to overfitting on the machine learning algorithms that these subsets of features are trained with. Filter methods does not have that tendency (Paul, 2020)

CHAPTER 3

PROPOSED FRAMEWORK

This chapter provides detailed information about the framework used in this thesis. As mentioned earlier, our goal is to find the best technique to achieve diversification while keeping relevance high. In this section, we describe our process in three sections. In the first section, we describe how the data is preprocessed. We explore different options for preprocessing and try to find the best approach for the library data. In Section 2, we describe how to categorize the library data, in order to provide a basis for diversification of the search results. In Section 3, we describe how the searching algorithm is implemented and how the diversification aspect is applied during this process.

3.1. PREPROCESSING

Preprocessing is an important step in most information retrieval tasks. Without proper preprocessing, the data would be extremely noisy. Also many algorithms would not be able to associate practically identical features. In general, the textual data must be converted into a numeric representation to be understood by machine learning algorithms. In the literature, this process is called feature engineering and it is even more significant for the textual data that are inherently unstructured (Tina, 2020).

We applied preprocessing in three steps. First, we format the textual data into a standard form. Second, we performed stemming techniques to obtain the roots of the words. Last, we apply feature elimination, to reduce the corpus size. We want to explore how the stemming and feature elimination changes the performance of our diversification algorithm. In this regard, our objective is to find the preprocessing approach that is best suited to our dataset and the library search problem.

3.1.1. Formatting

Formatting the textual data is an essential step for any text processing application. Textual data can contain many variations on words that make those words textually different, while they are essentially identical. Formatting the data into a standard form eliminates such issues. Most textual datasets include many unnecessary words such as stopwords, misspelling, punctuation etc. These words behave like the noise of the text data and had adverse effects on the performance of machine learning algorithm; especially statistics based algorithms. Several techniques are used to clean up such words. For example; numbers, punctuations, tags, stopwords, special characters can be removed, text can be converted to lowercase, lemmatization and stemming can be used etc. (Kowsari, Jafari Meimandi, Heidarysata, Mendu, Barnes, & Brown, 2019).

We performed the formatting of library data as follows;

- **Data concatenation:** The library data was available in Comma Separated Values (CSV) format. Each row in the spreadsheet represented a certain document. There are several columns in the spreadsheet; such as documents' physical location, title, author, Dewey codes, and publication year. Physical location of the document and the publication year are considered not useful information for the purpose of this study. Thus, we ignored those. Dewey codes are library cataloging specific codes that describe some attributes of the documents. Although these codes are not used in any searches, as they represent some document properties, they will be helpful for associating documents with each other. We used the text in the title, author, and Dewey code in order to represent a document.

- **Character-wise formatting:** We perform character-wise formatting on the title and author information of the document.

  We use Table 3.1 to change each character into its corresponding character. The majority of the documents in our library dataset are in Turkish. However, it is important to emphasize here that, in the document collection; there are also documents in other languages. Thus the character set of the document representations is restricted to Turkish and English lowercase letters and roman numerals. As can be seen from the table, we converted nuanced characters to the closest form that is in our character set and eliminated any other characters such as punctuation marks.

- **Dewey string formatting:** Dewey codes have to be in a certain format. They contain only numerals and can include a single '.' character. However, in our dataset, Dewey codes contain multiple library science-specific formats and older notations. Since correcting such codes, or converting them into a single unified version requires manual effort and is not trivial, we ignored any code that does not fit into the standard Dewey format. However, we also accepted older formats, such as which that starts with a 'T' or 't' character.

Table 3. 1:*Character map for character-wise formatting*

Current Character	**Corresponding Character**
c>='a' and c<='z'	Same character
c>='0' and c<='9'	Same character
c>='A' and c<='Z' , 'Ö', 'Ç', 'Ş', 'Ü', 'Ğ'	Lowercase character
'İ', 'í' , 'î' , 'Î' , 'ï' , 'ì' , 'Ì' , 'Í'	'i'
'â' , 'ä' , 'Â' , 'á' , 'à' , 'Ä' , 'æ' , 'Á' , 'å' , 'ã' , 'Å' , 'Æ'	'a'
'û' , 'ù' , 'Û' , 'Ú' , 'ú'	'u'
'é' , 'ê' , 'è' , 'ë' , 'É' , 'Ê' , 'È' , 'Ë'	'e'
'ó' , 'ô' , 'ò' , 'Ó' , 'Ô' , 'õ'	'o'
'ñ'	'n'
'ÿ'	'y'
'Đ'	'd'
'ß'	's'
Anything Else	Replace with space

- **Duplicate space elimination:** We eliminate any duplicate spaces between words and created the formatted text for each document.

3.1.2. Stemming

Information retrieval algorithms treat each word as a distinct entity representing an informational aspect of a document. However, that is not always the case. Two words that differ by a suffix can be strongly related to each other. Also, in many languages the suffixes may indicate a relation. As representing such complex relations between words are quite hard, studies on Natural Language Processing field try to alleviate this problem. However, most of these methods are language dependent. Thus many approaches treat each word as distinct features and try to discover relations through processing the data.

On the other hand, as a preprocessing step, we have some control about how to represent textual data. Mainly, stemming is a powerful natural language processing tool that has a potential to improve performance of the IR algorithms. During the preprocessing step, we might analyze each word in our dataset and extract the stem of the word, getting rid of all the suffixes. That way, the algorithms can associate different words stemming from the same root. That is a desired association in many situations. However, it is important to note that, as an end product of stemming, we would lose information contained within suffixes. Thus that is an undesired consequence, a tradeoff, of stemming.

Stemming has another problem of associating unrelated words. For example; Turkish word “çevrim” means “cycle, conversion, turning etc.”. However, it can be paired with another words “çevrimiçi, çevrimdışı” to mean “online, offline”. The stem of “çevrim” is “çevir” which means “to turn”. Accordingly, when we use stemming, IR algorithms would incorrectly associate any documents that talk about online, offline to other documents that talk about turning something.

Intuitively, we might think if our dataset is large enough, stemming might not be ideal. By that we do not mean the dataset being large just in the number of documents but the information amount in each document should be large as well. In that case, we might think there should be enough information in the documents, so that even un-stemmed words appear in so many documents, IR algorithms can discover hidden relations. In that case, not losing the information content of suffixes would be beneficial. However, for smaller datasets, stemming could be beneficial, since it allows IR algorithm to be able to find relations that it won't otherwise be able to find.

For library data, we decided describe what versions, not clear and try to find which performs better. Consequently, we generated two versions of the dataset. For one version, we use the original dataset. For the second version, we process each word in order to find its and replace that word with its stem. If we cannot find the stem of the word, we simple leave it as is.

For stemming, we use Zemberek NLP library for Turkish (Akın, 2019). Zemberek is an open-source library that can perform several NLP operations. We only used stemming function from the morphology module. With stemming, the unique feature count in the corpus is reduced by 16%.

3.1.3. Feature Reduction

Each unique word found in the corpus is represented as a feature. Accordingly, the total number of features is quite high. For our library dataset, we have 529,412 features. If we use stemming, we can reduce it to 444,559. Since the number of features determines the dimensionality of the vectors representing each document, higher number of features increases the computational overhead. Thus, eliminating some of these features and reducing the dimensionality would have good impact on the time requirements.

In order to run feature selection algorithms, we need to represent the documents in vector format rather than textual format. Bag of words model, in conjunction with TF-IDF models is a widely used alternative for this purpose. Bag of words model have an obvious drawback particularly for large corpuses: Some words are so common; they appear on most documents, such as “the”, “and” etc. Thus, they will overshadow other features by their overall weight and create unwanted statistical relations between documents. TF-IDF model tries to overcome this issue by using a modified metric for the weight of a feature, instead of just document frequency. TF indicates term frequency while IDF indicate inverse document frequency.TF-IDF model multiplies these factors. Term frequency measures the frequency of a word in a document; i.e. TF = total number of times word appears in a document / total number of words in that document. Inverse document frequency is the representation of how valuable a word is. It is an inverse fraction function that is logarithmically scaled. IDF gives low scores for words that appear in lots of documents (Tina, 2020).

$$IDF = \log \frac{number\ of\ documents}{number\ of\ documents\ word\ appears\ in} \tag{6}$$

Another issue about features is their varying information content. Some features are quite valuable in terms of their information content, while some features are practically worthless. For example, the word “diversification” is quite valuable. Existence of this feature in a document, together with a few other features can determine the topic of the document quite accurately. On the other hand, generic words like “the” or “and” can be found in most of the documents and they do not contain any meaningful information. Similarly, there can be some extremely rare words, or words containing typos, that appear only in one or two documents. Those features do have any real weight associating similar documents together. So they are not useful as well. Accordingly, in order to eliminate less valued features and keep high value features, we need to determine the information content of each feature.

Information gain is a tool in information retrieval toolbox that can be used to measure the information content of each feature. In this thesis, we used this algorithm for this purpose. The “mutual_info_classif” function implemented in “sklearn” library in Python is used in this thesis (Pedregosa, 2011). This function uses the dataset in TF-IDF format. We simply feed the entire corpus to this function, and it returns the value of each feature. We repeat this process for both regular corpus and stemmed corpus.

We want to experiment with different feature reduction ratios and see which one performs the best. Initially we were planning to leave a dataset without eliminating any feature as a baseline comparison. However, when we analyze the corpus and the outputs of info gain algorithm, we observed that about half of the features in our corpus appear in only one document. Accordingly, they are practically worthless for information retrieval. Thus, we can eliminate 50% of features and still perform the same as original dataset. With further analysis of the outputs of the info gain algorithm, we determined to analyze with feature reductions up to 50%, 30% and 10%, that is the percentage of feature left in the corpus after feature elimination.

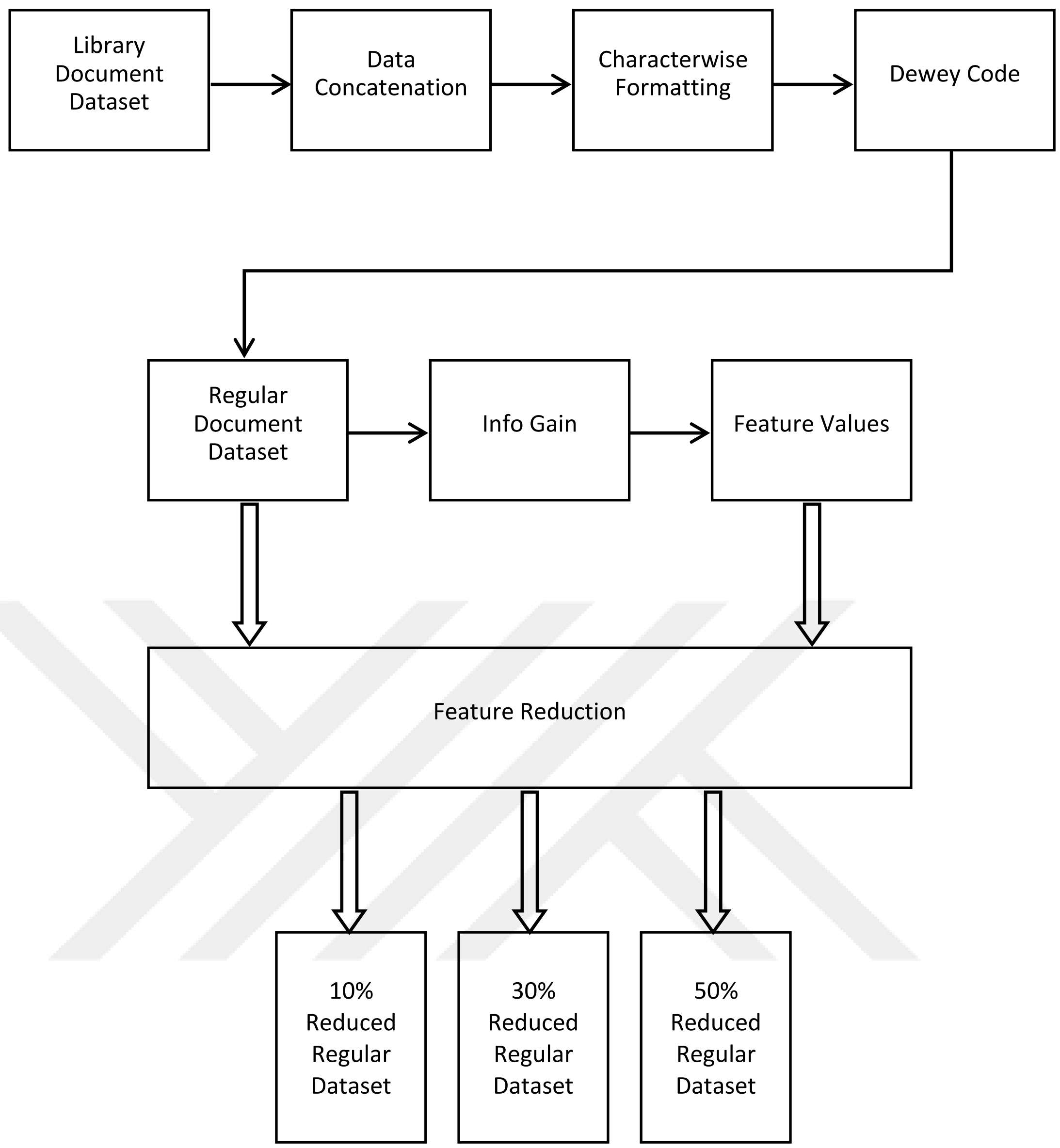

Figure 3.1: Preprocessing for regular datasets

For feature reduction, to create a dataset with k% feature count, we simply traverse our preprocessed corpus. For each feature we traverse, we check if it is in the top k% of features according to their TF-IDF values. If not, we simply eliminate it. In order to avoid eliminating all the features for a document, we implemented another constraint. If for a document, none of the features of this document appears in the top k% of the features, we keep the feature with the best value. This way we guarantee at least one feature remains for a document. This constraint increase the actual feature counts slightly, so we cannot reach our goal of k% feature count reduction. However, since the differences are not high, we continued the analysis with this approach.

We repeat the feature reduction for both regular and stemmed datasets. We used 3 feature reduction ratios; 10%, 30%, and 50%. Thus, after this step, we obtain 6 different datasets to analyze further.

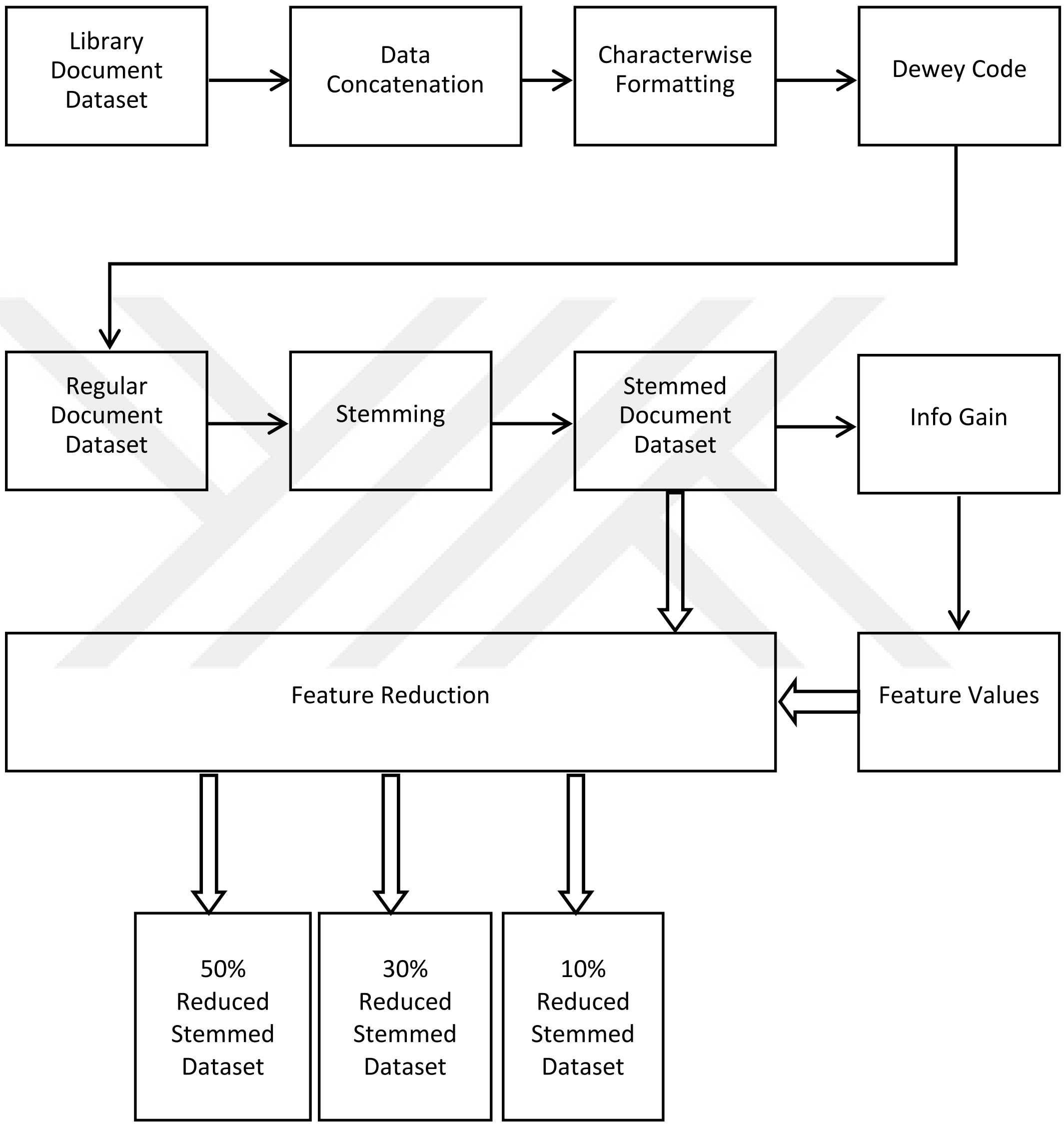

Figure 3.2: Preprocessing for stemmed datasets

3.1.4. Query Formatting

Library dataset contains a query dataset in addition to document datasets. Its formatting is done in a similar manner to document formatting but a little differently.

Library query data is also stored in a spreadsheet file. It has two columns. First column stores the query for search by keyword. Second column stores the query for search by author. Either of these columns can be full or empty. Similar to document formatting, we simply combine these columns into a single line.

Second, we perform the same character-wise formatting and duplicate space elimination as document formatting. Similar to document formatting, we created two query dataset, one with stemming and one without. We will use the stemmed query dataset with stemmed document datasets during the analysis phase. And we will do the same for regular datasets as well. We do not perform any feature elimination for query data.

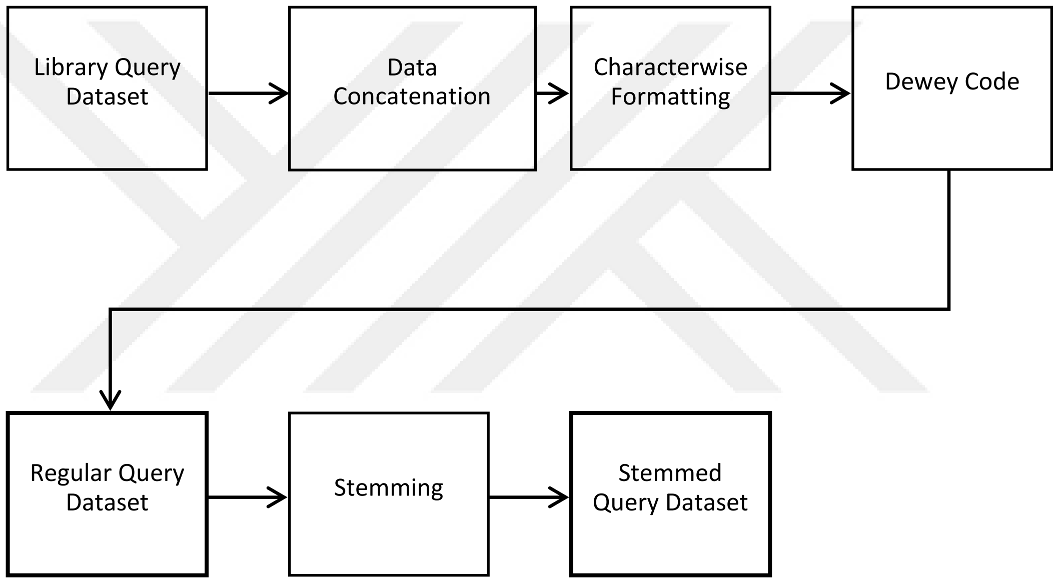

Figure 3.3: Preprocessing for query datasets

Figure 3.1 illustrates the overall preprocessing step for regular datasets. This process produces 3 datasets. Figure 3.2 illustrates the preprocessing process for stemmed datasets. This process also produces 3 dataset categories. After the preprocessing step we obtain 6 different datasets. Figure 3.3 illustrates the preprocessing done for the query datasets. This preprocessing step outputs, regular query dataset and the stemmed query dataset. After all the preprocessing, we created 6 document datasets and 2 query datasets. We also would like to note that, the original query log is used along with the orginal dataset, and stemmed query log is used along with the stemmed dataset in our experiments.

3.2. CLASSIFICATION OF DOCUMENTS

Classification of documents, generally lie within the topic modelling concept. Topic models are summarization techniques that are used to extract the topic of the documents. Each topic can be represented as a collection of features from the corpus. Topics represent concepts that are easily distinguishable from each other. However, usually overlapping topics are obtained. Topic models are quite useful when summarizing large corpuses of extracting key ideas from large datasets (Tina, 2020).

Different techniques are used for topic modeling. Most of them contain some form of matrix decomposition. One example is Latent Semantic Indexing (LSI). LSI uses Singular Value Decomposition (SVD). Another example is Latent Dirichlet Allocation (LDA). LDA (Latent Dirichlet Allocation) is an unsupervised machine-learning model that takes documents as input and finds K topics from N documents as output (note that, K is user-defined). It uses a generative probabilistic model. In this model, each document consists of a combination of several topics and word can be assigned to a single topic.

LDA (Latent Dirichlet Allocation) is an unsupervised machine learning model for soft classification. It gets a set of documents as input and finds K topics where, K is a user defined number, after processing the documents. Each topic is represented as a weighted list of features (Revert, 2018). An illustrative example given below,

*thesis * 0,2 | manuscript * 0,15 | novel * 0,09 |…*

LDA model has three main parameters; number of topics, number of features per topic and number of topics per document. LDA is a complex, hard to fine tune algorithm. Getting good results with LDA require extensive knowledge of the algorithm. The inner workings of LDA algorithm is outside the scope of this work (Revert, 2018).

Search result diversification algorithms try to select a set of documents belonging to different categories. In order to do that, we need to be able to know which category each document belongs to. There are several methods to do that. Related works section explains several approaches in more detail. For this work, we opt to use Latent Dirichlet Allocation (LDA) algorithm. This algorithm first proposed in the population genetics context (Pritchard, Stephens, & Donnelly, 2000), then used in machine learning field (Blei, Ng, & Jordan, 2003).This algorithm is an unsupervised clustering algorithm, so it does not require any external knowledge. LDA can classify documents into user selected number of categories. LDA does not require labeled data. It distributes the training data among user specified number of classes in an optimal way. Then the category of any new data can be predicted with this model as well.

We used “LatentDirichletAllocation” function implemented in “sklearn” library in Python (Pedregosa, 2011). Similar to other machine learning algorithms, this function is trained with real data. Then trained model is used to determine the category of any given data. LDA does not output a category id for given documents. It rather returns probabilities of given document belonging to each category. We then use these probabilities to determine the category of each document. LDA’s approach, computing the probabilities for each category, provides additional information about its prediction. For example, for some documents, let the probability of a category is much higher than the probabilities of other categories. Then we understand LDA is quite sure about its prediction. However, for some documents, let the probabilities be almost equal among categories. Then we understand, LDA is not sure at all about its prediction.

In our work, we used LDA a little differently. Since our focus is not to derive a category prediction model for library data, we did not divide our data as training data and experiment sets. We used all of our documents data for training. Our intention is simply assigning each document an implicitly generated category, so that we can continue with diversification analysis. Thus, using all the data for training would result in better classification for existing data. After training, we used the trained model for predicting categories for our documents. In this regard, we used LDA for clustering, rather than for classification. In our implementation, we did not examine how confident LDA was about its category prediction of documents. For each document, we assign the category with the highest probability to each document.

In our work, the symbol “C” represent the number of categories. We performed experiments with 4 different C values; 50, 100, 200, 300. We also repeat the same process for 6 datasets we have created in the previous section. Thus, we obtained a total of 24 datasets for our diversification experiments. Figure 3.4 illustrates the classification process for a single dataset. In this example, we used stemmed document dataset with 30% feature reduction. This process is repeated for all datasets.

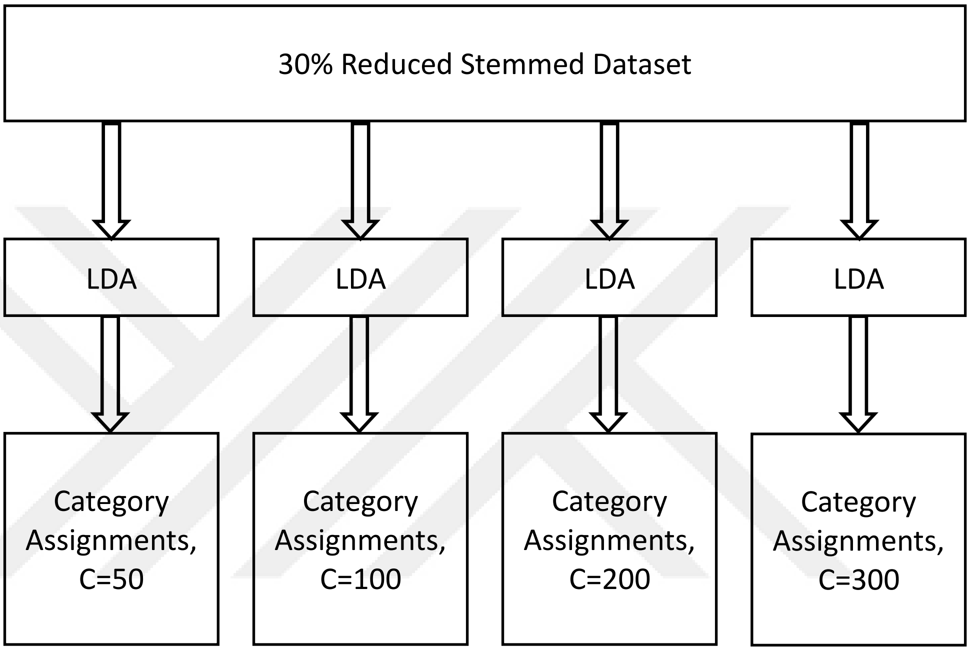

Figure 3.4: Classification for a single dataset

3.3. SEARCH RESULT DIVERSIFICATION ANALYSIS

The main goal of this work is to analyze how diversification approach works on real world library dataset. For this purpose, we implemented a MMR based search result diversification algorithm. In principle, our algorithm is quite similar to MMR. However, original MMR tries to handle diversity by selecting results that are as dissimilar to each other. This approach is computationally expensive. Instead we modified the definition diversity in MMR. Thus, we were able to reduce the computational overhead significantly. In this section, we will give details about the proposed algorithm.

3.3.1. Dataset Vectorization

There are some widely used vector representation alternatives for textual datasets. Bag of words model is a simple vector space representation model for unstructured text data. As the name implies, the bag of words representation does not retain word positioning in a document. Each document is represented with the terms, a sparse but very high dimensional vector, where each feature (a feature can be considered as a specific word in the corpus) is represented as a dimension in the vector. The values of the vector can be occurrence counts, binary appearance values, or some other metric. The grammatical relations and the order of the words are not preserved. The corpus of the documents consists of all the unique words (features), used in all the documents. The overall representation can be considered as very large sparse matrix where each document is represented as a row and each column represents a unique feature. Since representing this matrix as regular matrix format would require enormous amount of memory even for moderate size datasets, sparse matrix representations are often employed (Tina, 2020).

Another widely used vectorization method is Bag of N-Grams models. Bag of N-Grams model is an extension of Bag of Words model. The primary weakness of Bag of Words model is that it cannot hold any information regarding the order of words. Bag of N-Grams model is an attempt to improve the model on that front. Basically, instead of using each word as a feature, sequence of N words is used as features, preserving the order of words in a limited manned. If N is used as 2, then ordered pairs of words are used as features. If N is set to 3, then triplets of words are used as features. If high N values are used, the number of features in the vocabulary increases significantly, while frequency of each feature drops. At the same time, more information regarding the order of words can be represented. However, since the number of features increase and their frequencies drop, statistical analysis based machine learning algorithm will have harder time capturing inherent relations between features. They will need more data. Low N values lead to fewer features with higher frequencies, enabling machine learning algorithms to work more efficiently. However, they do not represent too much information regarding the word order (Sarkar, 2018) (Tina, 2020).

In this work, we used Bag of Words method for its simplicity. In our algorithm, we use a binary representation model. This choice allows the representation become simpler and faster to process.

```
vectorize(Documents, Queries)
   IDMap = AssignUniqueIDs(Documents, Queries)
   foreach document d in Documents, Queries
      foreach feature f in d
         Replace(f, IDMap[f])
      Sort(d)
      EliminateDuplicateFeatures(d)
```

Algorithm 3.1: Dataset Vectorization Algorithm

Algorithm 3.1 summarizes our vectorization process. In the first step, we take our all Documents and query data and assign a unique integer id for all features. Next, terms are sorted with respect to their occurrence count and each term is attributed to a numeric identifier.

Then we simply process every document in our preprocessed Library Document dataset and Library Query dataset. First, we replace each textual feature in the document with its integer id. Then we sort the integer features in increasing order. This will allow document-to-document comparisons faster, since we will just to traverse feature lists of each document just once.

3.3.2. Search Algorithm

Our search algorithm is inspired from the MMR algorithm. Our method is more similar to the swap approach described in (Vieira, et al., 2011), which also keep a candidate result set and swaps the documents that improve that candidate result set. Our algorithm is also partly related to local search approaches described in (Zuccon, Azzopardi, Zhang, & Wang, 2012) and (Naini, Altingovde, & Siberski, 2016). The algorithm aims to find k documents for a given query q, in a manner that maximizes the weighted linear combination of relevance scores and diversity scores of the selected documents. In order to do that, we should define relevance and diversity scores of selected result set. We define relevance score based on *cosine similarity* of each document and the query. We used the *average of cosine similarities* between each document in the selected result set and the query as the relevance score of the selected result set. The formulations of these calculations are provided in equations 7 and 8, respectively.

$$cosineSimilarity(d0, d1) = \frac{dotProduct(d0,d1)}{|d0||d1|} \tag{7}$$

$$relevanceScore(S, q) = \sum_{i=1}^{S.len} \frac{cosineSimilarity(S[i],q)}{S.len} \tag{8}$$

Cosine similarity is a widely used metric to define relevance between two documents. It is relatively cheap to compute and it is a normalized distance metric: Unrelated documents would have a 0 cosine similarity, while identical documents would have 1. We use the relevance score of a selected result set, as the average of the cosine similarity between each document in the result set and the query.

Cosine similarity represents the similarity between two documents as the cosine of the angle between vectorized multidimensional representations of these documents. Since vector representations of documents only contain non-negative values, the angle between any two document vectors can be in between 0^o and 90^o. 0^o can be obtained only if documents are identical; then the cosine similarity will be 1.0. 90^o can be obtained if two documents do not share any common feature, then the similarity would be 0.0. Cosine similarity can be computed using a cosine rule for dot product.

```
cosineSimilarity(v0 , v1)
   value = 0.0;
   i=0
   j=0
   while i <v0.len and j <v1.len
      if (v0[i] == v1[j])
         value += 1.0
         i++
         j++
      else if (v0[i] <v1[j])
         i++
      else j++
   value /= (sqrt(v0.len)*sqrt(v1.len));
   return value
```

Algorithm 3.2: Cosine Similarity Algorithm

The implementation detail of the cosine similarity depends on how the vectors are represented. In our implementation, since we are using a sparse vector representation, where the value can either be 0 or 1, we simply represent the vector as an integer array, that contains the id's of features found in the document. Also the integer arrays are sorted as explained earlier. Thus computing the dot product of such two vectors would be reduced to counting the common values of these vectors. The lengths of these vectors would also be reduced to taking the square root of the number of values in each array. Accordingly, the cosine similarity algorithm can be implemented quite efficiently. Algorithm 3.2 explains the cosine similarity computations more formally.

Diversity score is defined differently than standard MMR. Since we employ an implicit classification algorithm, LDA, and determine the category of each document, we can simply build upon this information. For diversification, we want selected search results to cover as much category as possible. For minimum, all the documents in the result set belong to the same category. For maximum, all the documents in the result set belong to different categories. Thus, we simply need to count how many different categories are represented by the search result and normalize it. Let our search to find top k documents. Then there can be at least 1 and at most k different categories found in the search results. If we normalize these values to [0,1] range, we can get the diversity score.

$$diversityScore(S) = \frac{numberOfDistinctCategoriesInS - 1}{\min\,(S.len, maxNumberOfDistinctCategories) - 1} \tag{9}$$

We included a correction to our diversification formula, with the min term to cover some exceptional cases. Usually, the number of total categories is higher than the number of documents we requested from the search. In that case, we did not need to use min term in the formula and simply use S.len instead. However, if we request more documents than the total number of categories than we would give lower diversification scores even though, the search returns the maximum diversification possible. For example, in our experiments for some experiment configurations, we used the number of categories as 50 and we searched for top 100 documents. In this case, if we have not included the min term in our formula, diversification scores would be at most 0.495. Since our goal is to normalize the scores to 0 for the worst possible diversification and 1 for the best possible diversification, including the min term to the formula makes sense.

$$combinedScore = \alpha * \mathrm{relevanceScore} + (1 - \alpha) * diversityScore \tag{10}$$

When we compute both relevance and diversity scores, we need to combine them to have an objective function to maximize. As explained earlier, we just use a weighted linear combination of these values. The weight parameter; α, is a hyper-parameter for our system and we are trying to find the best value for it as well.

```
findBestK(k, Documents, query, alpha)
   foreach document d in Documents
      d.RelevanceScore = cosineSimilarity(d, query)

   sortByRelevance(Documents)

   //Setting the most relevant k documents as initial result set
   resultSet = Documents[0, k]
   relevanceScore = getRelevanceScore(resultSet)
   diversityScore = getDivesityScore(resultSet)
   combinedScore = relevanceScore* alpha + diversityScore* (1 - alpha)

   for i = k upto Documents.len
      if d.category is in resultSet.categories
         continue

      candResultSet = resultSet
      d = findLeastValuableResult(candResultSet)
      //least valuable result is the one with smallest relevanceScore,
      //whose category is represented in ResultSet at least twice

      candResultSet.replace(d, Documents[i])
      candRelevanceScore = getRelevanceScore(candResultSet)
      candDiversityScore = getDivesityScore(candResultSet)
      candCombinedScore = (candRelevanceScore* alpha) +
                          (candDiversityScore* (1 - alpha))

      if candCombinedScore > combinedScore
        //If new document improves combined score, update the resultSet
         resultSet = candResultSet
         relevanceScore = candRelevanceScore
         diversityScore = candDiversityScore
         combinedScore = candCombinedScore
         if diversityScore == 1.0
            break
         //If we reach maximum diversity score, no need to check less
         //relevant documents any more.
      else break
      //If the new document cannot improve the combined score, then
      //no other documents can improve, since they cannot improve
      //diversity score more than this document, but their Relevance
      //score will be even lower. So we just break out of the loop
   return resultSet
```

Algorithm 3.3: Search Algorithm to find top k documents for a given query

Up to this point we described the score for how to find the scores for a selected subset of documents. However, finding the best subset can be a relatively hard problem. Checking every possible subset is not feasible. In our work, we employed a search algorithm that finds the best subset relatively fast. It was made possible by our definition of relevance and diversity scores. There are other techniques such as inverted indexes for document query matching. Inverted indices technique is quite common and very efficient. Cambazoglu et al. describes efficient parallelization of inverted indices (Cambazoglu, Kayaaslan, Jonassen, & Aykanat, 2013). However we decided not to use inverted indices to keep our implementation simpler. Algorithm 3.3, describes the details of search algorithm.

The search algorithm finds the user-specified k best results for given query on Library Document dataset. It also takes an additional argument named “alpha”, which is the weight parameter in order to control the balance between relevance and diversity scores. It finds the best k documents that maximize the combined score and returns this k documents. Algorithm starts by computing the relevance scores between each document in the documents dataset and the query. For relevance score, cosine similarity is used. After relevance scores are computed, the Documents dataset is sorted in descending order, according to computed relevance scores. This part requires the majority of computational workload, since it has to work on all the documents dataset. On the other hand, relevance score computations can easily be parallelized. In our implementation, we used OpenMP to perform these operations using multiple CPU cores. That speeds up the search process significantly.

After the documents’ relevance scores are computed and the documents are sorted accordingly, we define our result set as the top k documents. Then we compute the current relevance, diversity and combined scores of this result set. Then we enter into a loop that iterates over the rest of the documents, to look for any improvements of result sets we can make. Since our current list contains the best possible records in terms of relevance, any improvement we can make will be via diversity improvements. Our diversity score is based on how many different categories are represented in the result set. Thus, any new candidate document, which has a category already present in the current result set cannot improve diversity score. So, we can just ignore such documents.

If the new candidate has a category that is not represented in the current result set, it can improve the diversity score. However, as it also has lower relevance score relative to other documents in the current result set, combined score may not improve. In order to achieve largest improvements, we should find the least valuable document in the current set. The least valuable document would be the one with smallest relevance score, whose category appears in the current result set at least twice. So removal of this record will not reduce the diversity score. Finding the least valuable document is done via the findLeastValuableResult function. This function traverses the candidate result set in reverse order. For each document, it checks if that documents category is identical with the category of another document further up on the candidate result set. When the function finds such document, it terminates the traversal and returns that document. Then we replace this least valuable current document with candidate document and compute the scores. If the scores improve, this means the diversity score improvements are higher than relevance score losses. Thus we accept this change and continue.

However, if the candidate document cannot improve the scores, we conclude that our current result set is the best candidate and stop the search process. The reasoning behind this conclusion relies on our diversity score definitions: Our diversity scores improve in a stepwise fashion. So if after a document replacement, diversity scores increase *x* amount, then after the next document replacement it will increase *x* amount again. If replacing current least valuable document in the result set with the candidate is not improving combined score, replacing the next least valuable document would be even worse. More formally, let *l* be the least valuable document and let *c* be the candidate document. Then the change of combined score upon the replacement of *l* with *c* can be described as;

$$CombineScoreChange = \frac{\alpha * (\mathrm{c}.relevanceScore - l.relevanceScore)}{k} + \frac{1-\alpha}{k-1} \tag{11}$$

The first part of the equation describes the average relevance score change times. The second part represents the diversity score change times 1-α. The diversity score change is due to the increased number of categories represented in the result set.

Let l_0 be the least valuable document in the result set and let l_1 be the next least valuable document. Let c_0 be the current candidate document and let c_1 be the next candidate document. Let us compute the combined score changes if we replace these candidates with their counterpart least valuable documents.

$$CombineScoreChange(c_0, l_0) = \frac{\alpha * (c_0.relevanceScore - l_0.relevanceScore)}{k} + \frac{1-\alpha}{k-1} \tag{12}$$

$$CombineScoreChange(c_1, l_1) = \frac{\alpha * (c_1.relevanceScore - l_1.relevanceScore)}{k} + \frac{1-\alpha}{k-1} \tag{13}$$

$$CombineScoreChange(c_1, l_1) - CombineScoreChange(c_0, l_0) =$$

$$\frac{\alpha}{k} * ((c_1.relevanceScore - l_1.relevanceScore) - (c_0.relevanceScore - l_0.relevanceScore)) =$$

$$\frac{\alpha}{k} * \big((c_1.relevanceScore - c_0.relevanceScore) + (l_0.relevanceScore - l_1.relevanceScore)\big) \tag{14}$$

By definition l_1*.relevanceScore>=* l_0*.relevanceScore* and c_0*.relevanceScore >=* c_1*.relevanceScore*. Thus, since *alpha* and *k* values are always positive, the $CombineScoreChange(c_1, l_1) - CombineScoreChange(c_0, l_0)$value will always be less than or equal to 0.

So, $CombineScoreChange(c_1, l_1) \leq CombineScoreChange(c_0, l_0)$ inequality will be true. Accordingly, once we reach to a candidate and its least valuable counterpart, that does not improve the combined score of the result set, we can safely say no further candidate can improve the combined score as well. Thus we can safely break out of the loop.

In our algorithm, if after a replacement with a candidate, we reach a diversity score of 1, we also break out of the loop. Because, this means we have reached maximum diversity and it cannot further improve the results. In this case, the search can terminate ensuring no further improvement is possible.

The asymptotic complexity of the algorithm is O(Nlog(N)) where N is the number of documents, since the process is mostly linear, but sorting the document dominates the complexity. The computation of relevance scores of documents is linear. In the worst case, the loop that tries to find the optimal result set has to scan all the documents. However, most probably, the algorithm would break out much sooner. For sorting part, we used comparison based quicksort algorithm. Thus, we stated that the complexity of our algorithm is O(Nlog(N)). On the other hand, if we analyze the sorting part, we can see since the number of documents is in millions and we are sorting with respect to a floating point value, we can easily employ a linear time sorting algorithm, like radix sort. Accordingly, we can easily reduce the asymptotic complexity to O(N). However, for our workload, the actual runtime difference between a radix sort implementation and a quick sort implementation does not differ much. Hence, we used the more generic quick sort implementation.

3.4. ON THE EVALUATION OF THE EXPERIMENTAL RESULTS

Our work has two main goals. First one is to observe how the diversification approach works on the library dataset. Second, analyzing the effects of different approaches and hyper-parameters we tried on the diversification. In order to do that, we need to be able to quantify the overall performance of the search algorithm, with different parameters. Fortunately, our library dataset contains real user queries. Thus, our experiments will represent the real world performance. In our analysis, we simply perform the search operation described in the Section 3.3, for each query in our dataset. Then we calculate the mean values of relevance scores, diversity scores and combined scores for all the query searches. As explained in section 3.1, with stemming or not stemming, with three different feature reduction ratios and with four different number of categories of classification we have 24 different datasets. Also, as described in Section 3.3, the search algorithm takes two more parameters; *k* and *alpha*. *Alpha* represents the weight between relevance and diversity. In our implementation, higher values favor relevance. Finding the optimal *alpha* value is quite important for the real world performance of the diversified search algorithm. Thus we tried several values. After some playing around with some parameters, we observed that too low *alpha* values will not be favorable, since the Relevance would drop significantly. So we did not try *alpha* values lower than 0.5 in our formal experiments. However, we included *alpha* equals 0.0 case for comparison purposes. We tried 12 different *alpha* values; 0.0, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1.0.

The search algorithm also takes *k* as another parameter. We believe, the search algorithm will behave similarly for different *k* values. However, we tried 2 different *k* values; 10 and 100 in order to verify. This means we searched for top 10 and top 100 documents for each query and analyzed the results. The overall analysis algorithm is given in Algorithm 3.4

```
analyze()
   for k in (10, 100):
      for stemming in (on, off):
         queryDataset=getQueryDataset(stemming)
         for featureReductionRatio in (0.1, 0.3, 0.5):
            for C in (50, 100, 200, 300):
               documentDataset = getDocumentDataset(stemming,
                                 featureReductionRatio,C)
               for alpha in (0.0, 0.5, 0.55, …, 0.95, 1.0):
                  relevanceScores=0
                  diversityScores=0
                  combinedScores=0

                  foreach query q in queryDataset:
                     scores = findBestK(k, documentDataset, q, alpha)
                     relevanceScores+=scores.relevanceScore
                     diversityScores+=scores.diversityScore
                     combinedScores+=scores.combinedScore

                  relevanceScores/=queryDataset.length
                  diversityScores/=queryDataset.length
                  combinedScores/=queryDataset.length

                  print(k, stemming, featureReductionRatio, C,alpha,
                        relevanceScores,diversityScores, combinedScores)
```

Algorithm 3.4: Analysis algorithm

We performed the searches of each query in the query dataset for every configuration we have. Thus, for 24 document datasets, 12 *alpha* values and 2 *k* values, we got 576 different set of experiments. We gathered average relevance, diversity and combined scores of all the searches made on the query dataset for each experiment set. We evaluated each decision we made individually. The detailed analysis will be given in the Results chapter. The table of raw values we got from these 576 experiment sets is given in Appendix 1.

CHAPTER 4

RESULTS

The goal of this work is improving the diversity without losing relevance. To this end we tried several approaches and tested them on real world library data. We want to see, if they will improve the diversity well enough without losing significant relevance, or if they will fail. Secondly, we want to find the best approaches and tune the hyper-parameters we used in our algorithm so the diversification performs the best.

We performed a total of 576 experiments, and in each experiment; we searched for each query in our query dataset consisting of 166529 real world user queries. We gathered all the results and analyzed them for each algorithmic decision or hyper-parameter. In the following sections, we will individually examine the effects of stemming, feature reduction ratio, number of categories for classification and the alpha value that is the weighting constant between relevance and diversity. The experiment parameters and ranges of these parameters are given in Table 4.1. Then we will try to determine the best possible approaches and hyper-parameters. And finally we will examine, the relevance – diversity tradeoff for real library dataset.

Table 4. 1: *Experiment parameters and ranges*

Experiment parameter	**Parameter Range**
Number of returned results (k value)	10, 100
Stemming	On, Off
Feature reduction ratio	0.1, 0.3, 0.5
Number of categories for classification (c value)	50, 100, 200, 300
Relevance diversity weighting constant (alpha value)	0.0, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1.0

In order to differentiate different algorithm choices and hyper-parameters in our experiments we used a naming convention. We represented the test results as follows; T[k,s,f,c,a], where k is the index for k number, s is the index for stemming approach, f is the index for feature reduction ratio, c is the index for number of categories for classification and a is the index for the alpha value. For example, T[10,1,0.1,100,0.75] represent the experiment with k value of 10, stemming is on, feature reduction ratio of 10%, C value of 100 and alpha value of 0.75. With this notation we can access the relevance and diversity scores of each experiment and compare different experiments differing by only one index. Although we try to maximize the combined score, it is not ideal to use it during analysis, since it is a hyper-parameter dependent variable.

When we analyzed the experiment results, we observed some unexpected results for the C parameters. Naturally, we expect higher C values to have higher diversification scores. Getting 10 or 100 distinct categories is expected to be much easier if there are 300 categories instead of 100. However, in our experiments we observed that the diversification scores for 300 categories is much lower. When we investigate the reason for this anomaly, we found out even though we set for 300 categories, only a fraction of categories were heavily assigned to most documents. Depending on the experiment configuration, 51, 75, 90, 121 or 129 categories are assigned to some documents instead of full 300. Hence, the diversity scores for these experiments were worse.

We analyzed other category parameters. For 50, 100 and 200 parameters, we observed that all the categories are assigned to some documents; i.e. the actual number of categories is indeed what is stated. For 300 case, the culprit for this issue lies within our category assignment strategy for documents. As explained in Chapter 3, Section 2 the classification algorithm we used, LDA, returns a list of probabilities for each document to be in each category. Then for each document, we selected the category with the highest probability. With this approach, a category may not be the highest probable category for any document and thus stay unassigned.

As a future work, we might try to change category assignment strategy along with diversity scoring strategy. This way, multiple categories can be assigned to each document along with their probabilities and diversity score would represent a more complex picture of actual diversity. In that case, 300 categories can be expected to work fine.

For our further analysis, we decided to leave the experiments with 300 categories out of the analysis. Since the diversity scores do not represent the actual value, these results would not be useful for comparison. Thus leaving them out will help to get a better picture of the effects of hyper-parameters. We also leave out the experiments with alpha values 0.0 and 1.0, since they will not consider relevance and diversity respectively. We included those experiments mainly for our alpha analysis and general performance analysis. They won't be as useful for hyper-parameter tuning. Thus, we will use, 360 set of experiments for our further analysis. In the following sections we will analyze each parameter of our framework. ECV stands for "Experiment Configuration Values", and represent every possible combination of parameters we used in our experiments.

The experiments are performed on a laptop with Intel i7 9750h CPU and 16 GB's of RAM. Windows 10 operating system was used. The "mutual_info_classif" function, used in the feature reduction computations, is executed twice; one for stemmed dataset and one for regular dataset. This function is implemented in sklearn library in python and it is not an especially fast implementation. We spent 24-25 hours of computation time per execution of this function. Thus in total feature reduction computations took about 2 days of computation. We had 6 different datasets to classify, 2 sets for stemming and 3 sets for feature reduction ratios. We decided to try 4 different number of categories for each of these datasets. Accordingly, we needed to perform the classification 24 times. We used the python function "LatentDirichletAllocation", implemented in sklearn library. Each of these classification tasks took about 45-60 minutes of computation time. Accordingly, in total we spent close to a day for classification computations. Other parts of the preprocessing took negligible time compared to feature reduction and classification workloads. So in total we spent about 3 days of computation for preprocessing steps.

We perform the experiments using a code we implemented in C++ language. We tried to implement the code as efficient as possible. We performed a total of 576 experiments. At each experiment, we searched for each query and analyzed the results. In total, all the experiments took less than 10 hours. This means a single experiment takes about a minute. Since we have more than 166000 queries in a single experiment, we can answer more than 2700 queries per second on a laptop.

This performance is more than enough for the search workload of National Library of Turkey.

4.1. EFFECTS OF STEMMING

We performed each experiment one with using stemmed words and one with regular words. Many machine learning algorithms, ours included, treat each word as a unique entity if the wording is different. Accordingly, a word and its many other words derived from this word via some suffixes are treated as if they are completely unrelated. When we use stemming, all the derived words will be reduced to the root word, so they can be treated as the same word. This will help to form associations between documents that contain different derivations of the same root word. However, since stemming removes the suffixes, it also removes potentially useful information. Thus, it may also hurt the performance of the algorithms. Thus, the effect of stemming can be positive or negative. We need to see from the experimental results.

In order to analyze the effect of stemming, we computed the following values;

$$\forall_{k,f,c,a\in(TCV)}\left(\frac{T[k,1,f,c,a].relevance}{T[k,0,f,c,a].relevance},\frac{T[k,1,f,c,a].diversity}{T[k,0,f,c,a].diversity}\right) \quad (15)$$

In essence, for each experiment configuration, we took the relevance and diversity scores of the configuration that use stemming and divide it to the one that does not. This way we can see, if the use of stemming improves the results or not.

Table 4. 2*: Results of stemmed compared to regular words*

Stemming wrt. Regular	Score Ratio	
	Relevance	Diversity
Min	1.0498	0.9552
Max	1.1253	1.0144
Average	1.07582	0.9998

The results given in Table 4.2 show a very clear benefit on using stemming approach for relevance. Even in the worst case, relevance scores improve more than

4.98%. On average, the relevance score improvement is 7.58% while it can reach more than 12% for the best case. For diversity scores the results are not as good. For the worst case we observe almost 4.5% reduction in diversity scores, while on average we observe almost identical scores. For the best cases we observe about 1.44% improvement. However, in general we observe that the diversity scores do not seem like to improve with stemming.

In general, we observe a clear and significant benefit for relevance scores. For diversity scores, for worst and best cases, we observe a little variation, but the average case results are almost identical. Thus, we can say stemming is beneficial for the overall algorithm.

4.2. EFFECTS OF FEATURE REDUCTION

In our experiments, we used three different feature reduction ratios; 0.5, 0.3, 0.1. Elimination of features can help the performance, since it increases the importance of remaining features. So if too many unrelated features remain in the dataset, the importance of each feature will be diluted. However, if we eliminate features that can potentially associated some documents, we may also hurt the performance. Thus the results of the experiments are important to determine the best ratio of feature reduction.

$$\forall_{f \in (0.1, 0.3)} \forall_{k,s,c,a \in (TCV)} \left(\frac{T[k,s,f,c,a].relevance}{T[k,s,0.5,c,a].relevance}, \frac{T[k,s,f,c,a].diversity}{T[k,s,0.5,c,a].diversity} \right) \tag{16}$$

We used a similar formula to the one we used in previous section. We take the feature reduction ratio of 0.5 as our basis and compare other ratios with respect to it. The results are as follows;

We observe differences just over 2% at the most. On average, we observe almost identical results. For diversity, we observe a similar variation. We see just over 2% difference at the most, but on average the results are almost identical. Even though we do not think the difference is significant, we can still argue that the 0.1 feature reduction ratio gave the best results.

__Table 4. 3__: Results of feature reduction ratios 0.1 and 0.3 compared to the feature reduction ratio of 0.5

Feature Reduction Ratios	0.1 wrt. 0.5		0.3 wrt.0.5	
	Relevance	Diversity	Relevance	Diversity
Min	1.0000	1.0000	0.9887	0.9951
Max	1.0223	1.0236	1.0214	1.0234
Average	1.0047	1.0037	1.0031	1.0023

Accordingly, we can conclude that the feature reduction ratio does not affect the performance of the algorithm significantly, for the ratio's we use. If we reduce the number of features to even lower ratios, we certainly would see some performance loss after some point. However, the feature reduction ratio of 0.1 performs perfectly fine. Thus it is quite preferable, since it reduces the dataset size as well.

4.3. EFFECTS OF NUMBER OF CATEGORIES FOR CLASSIFICATION

In this work we quantify the diversification based on how many different categories are represented within our search result. We used a LDA as a clustering algorithm to determine the categories of documents in the document collection. We also experimented with the number of categories, C, produced by LDA: If we set the C value too low, then by the nature of clustering, we would have to assign relatively unrelated documents to the same group. This will lead to an underestimation of the diversification in our results. On the other hand, if we set the number of clusters too high, then many closely related documents would have their own categories. Thus, their relatedness will not be measured by our diversification metrics. Consequently, we will overestimate the diversification in our results. In our experiments, we selected modest C values. These values are partly limited by the performance limitations of the underlying LDA algorithm, i.e. the LDA implementation we used crushed for higher C values. On the other hand, we also would not have wanted to go much higher either.

We define the diversity based on how many different categories can be represented with our search result. At worst, a single category will be represented by all the documents in the result set. At best, each document will belong to a different class. We formulate a simple scoring mechanism that gives 0.0 for the worst case and 1.0 for the best case. This approach works fine for k=10 cases with our selected C values. However, for k=100 and the C=50 case, we can get at most 50 different categories in the search results, not every document can belong to a different category as postulated earlier. We modified our diversification score to give the score of 1.0 to the maximum possible number of categories that can be represented with our experiment configuration. In other words, for k=100 and the C=50 case, representing 50 categories within the result set would get full score.

Determining the optimal C value is not a trivial task. The analysis we have made in the previous sections would not be as effective since the diversification scores are directly based on the C parameter. Intuitively, we can assume higher C values will lead to better diversification results, by simply statistical effects. However, the diversification scoring approach we used has a different effect for k=100 case. For k=100, reaching the maximum possible diversity is much easier for C=50 than for C=100 case. Accordingly, we can expect the scores would be much better for C=50 case for k=100. We perform the analysis despite the said drawbacks. For this analysis, we used a similar formula to the feature reduction ratio analysis.

$$\forall_{c\in(100,200)}\forall_{k,s,f,a\in(TCV)}\left(\frac{T[k,s,f,c,a].relevance}{T[k,s,f,50,a].relevance},\frac{T[k,s,f,c,a].diversity}{T[k,s,f,50,a].diversity}\right) \tag{17}$$

Table 4. 4: *Results of C=100 and C=200 experiments compared to C=50 experiments*

Score Ratio	C=100 wrt. C=50		C=200 wrt. C=50	
	Relevance	Diversity	Relevance	Diversity
Average All Experiments	0.9452	0.9534	0.9787	0.9840
Average, k=10	1.0106	1.0099	1.0252	1.0252
Average, k=100	0.8798	0.8970	0.9322	0.9427

Table 4.4 displays the average relevance and diversity scores for different C values with respect to C=50, for all experiments (k=10 or 100) case, k=0 case and

k=100 case individually. For k=0 case, we observe a slight improvement with increasing C value. However, for k=100 case, we observe a completely different result. The scoring system allow 50 categories case to shine through, since obtaining 50 or close to 50 categories with 100 documents is much easier than for 100 or 200 categories.

For the reasons described above and demonstrated through the average experiment results, the diversity scores are not quite comparable with each other. As a matter of fact, the numerical values of experiment cases with different C values are not comparable too. Accordingly, in order to quantify and compare results we used a different approach. For each experiment case, we might try to analyze the improvements over diversity scores compared to the same experiment configuration with alpha=1.0 value. This way we can estimate the effects of the C values normalized with respect to the case that does not take diversity into the consideration. Accordingly, we can analyze the C parameter in a more objective manner.

Table 4. 5*: Results number of average relevance scores and average diversity score increase compared to maximum relevance approach, for different C and k values.*

	Average Relevance Score	Average Diversity Score Increase w.r.t. α=1.0 case
C=50, All Experiments	0.3426	1.6720
C=100, All Experiments	0.3252	1.7568
C=200, All Experiments	0.3368	1.6225
C=50, k=10	0.3894	1.7001
C=100, k=10	0.3938	1.6003
C=200, k=10	0.4000	1.5115
C=50, k=100	0.2959	1.6432
C=100, k=100	0.2565	1.9133
C=200, k=100	0.2736	1.7336

Table 4.5 shows the results for the analysis described earlier. For k=10 case, C=50 improve the best, next C=100 and the least C=200. For k=100 case, C=100 improve the best, C=200 improve next and C=50 improve the worst. On average, C=100 improve best, C=50 improve next and C=200 improve the worst. However, for most cases, if the improvement is high, the relevance scores are lower. That is not true for k=100 case and for 200 categories, while it gives worse results for both diversity improvement and the relevance. In general, we observe some variations, but the differences in both average relevance scores and diversity improvements are small.

The C value is a very hard parameter to optimize. When we approach the data from different aspects we can make different analysis and get different results. Our first analysis, that are based on actual diversity scores show a preference towards 50 categories particularly for k=100 case and a small preference for 200 categories for k=10 case. However, 100 categories show the best improvement in diversity, as demonstrated in our second analysis. We believe, the optimal value for this parameter should be determined based on the user experience. We think users will exhibit a preference towards a more general versus a more detailed diversification strategy. However, we also believe, the importance of this parameter will not be too much, as long as we use a reasonable C value as we used in this work. For current work we think we should revisit tuning this value, after we narrowed our parameter space a bit more. So we can focus our analysis on the experiment results of more appropriate experiment configurations.

4.4. BALANCING RELEVANCE AND DIVERSITY

Selection of alpha value is quite important for getting a good balance between relevance and diversity. In this work, we tried 12 different alpha values. 2 of them, 0.0 and 1.0 are included just for reference. Intuitively we need to have a certain balance between relevance and diversity. So extreme alpha values are unlikely candidates to be optimal values. After a few quick experiments, we concluded that low alpha values, the values that favor diversity, does not work well. So we only experimented with 0.5 and more in addition to 0.0. The correctness of this assumption will be revealed after the experiments. If lower end of alpha values were more optimal, then we would see the 0.5 value would be the best value we have experimented. Thus, with our assumption, we did not lose any potential alpha value.

Our scoring strategy normalizes both relevance and diversity scores to [0,1] range. However, if we analyze deeper, we will see the relevance scores are much lower on average than divergence scores. In fact, this is quite expected. In order to have very high relevance score, the query and the document should match pretty closely. This might happen for a few documents, but since we are returning 10 or 100 documents, the average relevancies drop significantly. On the other hand, the divergence scores only require gathering documents from different categories. That is actually quite easy to achieve. Even for the experiments with alpha value is 1.0, the experiments that we selected top k relevant documents and do not care about the diversity at all, the average relevance score is 0.3781, while divergence score is 0.5700. This divergence score is the score for natural distribution of documents to categories. For all the experiments, the average relevance score is 0.3405 while the average divergence score is 0.9038. These results show us that in reality, the scales of our relevance and divergence scores are different. So the alpha values should be selected so this imbalance can be corrected. We can make a very rough estimate and compute the alpha value that gives equal importance to relevance and divergence as follows; $0.3406\,\alpha = 0.9038\,(1-\alpha)$, by using the average relevance and divergence scores. The *alpha* value can be computed from this equation as 0.7263. On average, this value seems to give equal importance to relevance and divergence. Of course we can favor one or the other, but we expect the optimal *alpha* value will be close to this value.

We also used another approach to determine the optimal alpha value. We included 0.0 and 1.0 *alpha* values as reference points to our experiments. *Alpha* equals 1.0 means that we select the top k relevant documents and did not care about diversity at all. Alpha 0.0 means we tried to maximize the diversity at all costs. Accordingly, for any experiment configuration, the relevance scores will be the worst at *alpha*=0.0 and will be the best at *alpha*=1.0. Similarly, diversity scores will be the worst at *alpha*=1.0 and will be the best at *alpha*=0.0. These best and worst scores can be considered the characteristic distribution of relevance and diversity for our dataset. Since we want to fairly weigh relevance and diversity, it may be ideal to normalize these scores within their characteristic distribution. So our scales of two scores will be on equal footing. The equation we used for this purpose is as follows.

$$\forall_{a\in(0.5..0.95)}\forall_{k,s,f,c\in(TCV)}\left(\frac{T[k,s,f,c,a].relevance-T[k,s,f,c,0.0].relevance}{T[k,s,f,c,1.0].relevance-T[k,s,f,c,0.0].relevance}\right) \tag{18}$$

$$\forall_{a\in(0.5..0.95)}\forall_{k,s,f,c\in(TCV)}\left(\frac{T[k,s,f,c,a].diversity-T[k,s,f,c,1.0].diversity}{T[k,s,f,c,0.0].diversity-T[k,s,f,c,1.0].diversity}\right) \tag{19}$$

According to this normalized metric, as *alpha* goes from 0.0 to 1.0, relevance score will also go from 0.0 to 1.0. Diversity score, on the other hand will go from 1.0 to 0.0. Accordingly, at some point they will intersect. We can think of that intersection point as the equal weighting *alpha* point.

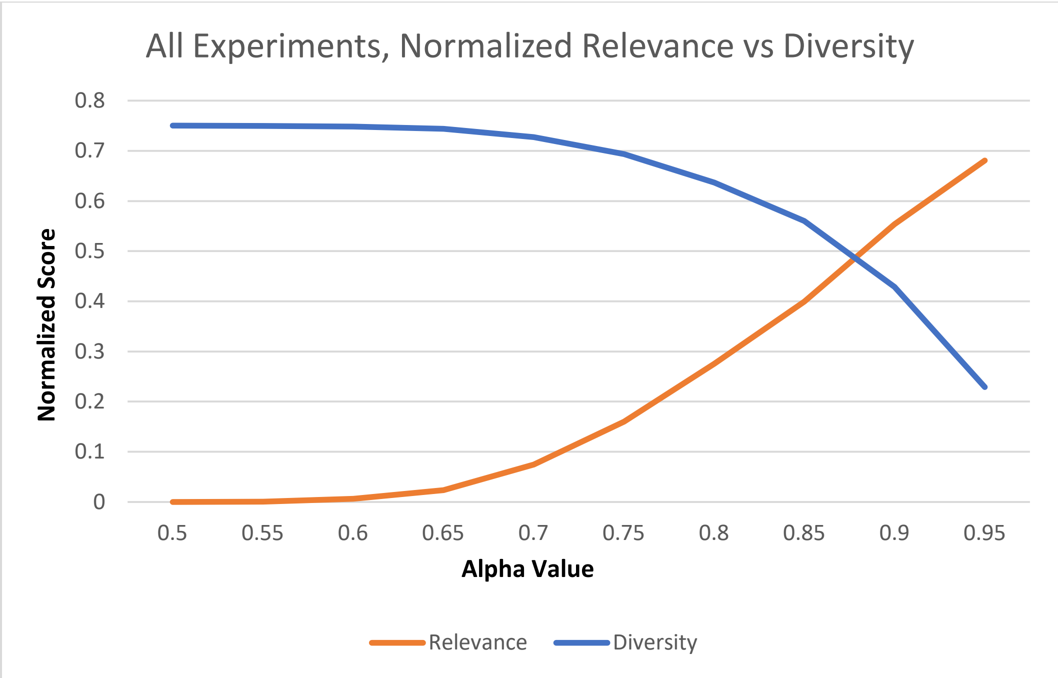

Figure 4.1: Normalized relevance and diversity scores with respect to the alpha value

Figure 4.1 shows the intersection point is between 0.85 and 0.9. If we compute the exact value, we would find the intersection point as 0.8782. Since, we only experimented *alpha* values with 0.05 intervals, the intermediate values in the graph are produced by linear interpolations. Thus, the intersection point we have found is actually an estimate and can differ. However, the normalized values relevance and diversity functions with respect to *alpha* seem to be increasing and decreasing functions respectively. They also seem to follow a relatively calm trajectory. Thus, our estimate can be off, but it will be close.

We also analyzed if the k value differs the intersection point significantly. Figure 4.2 and Figure 4.3 shows the intersection points for k values 10 and 100 respectively. We observe no significant difference at intersection point values for different k values. For k=10, exact intersection point can be interpolated as 0.8766 while for k=100 it can be found as 0.8799.

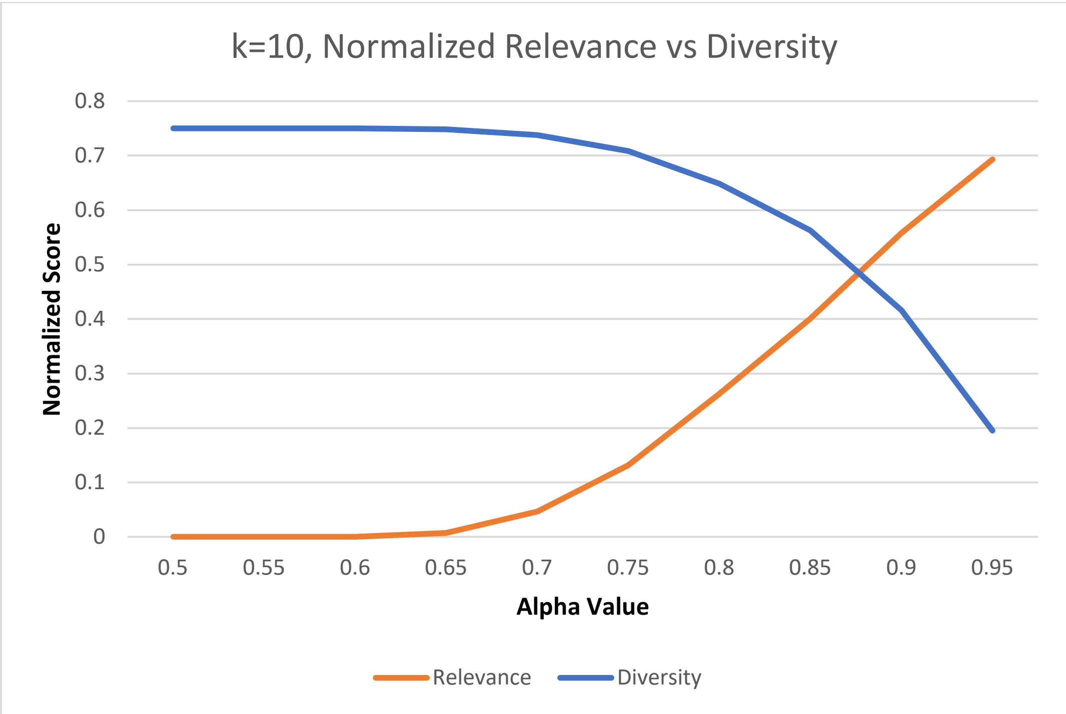

Figure 4.2: Normalized relevance and diversity scores with respect to the alpha alue for k=10

With two different approaches, we estimated the optimal *alpha* values as 0.7263 and 0.8782. These two approaches gave significantly different results. The main reason for this is the difference between natural scales of relevance scores and diversity scores. The relevance scores have much narrower range than diversity scores. Accordingly, when we use the first approach, larger range of diversity scores overwhelm the minor improvements of relevance scores. On the other hand, if we normalize the scores within their natural ranges, we get the opposite effect and small improvements of relevance scores are given too much weight. In practice, the optimal *alpha* value will probably be in between these estimates. We believe, the best way to fine tune the *alpha* value should rely on user experiences.

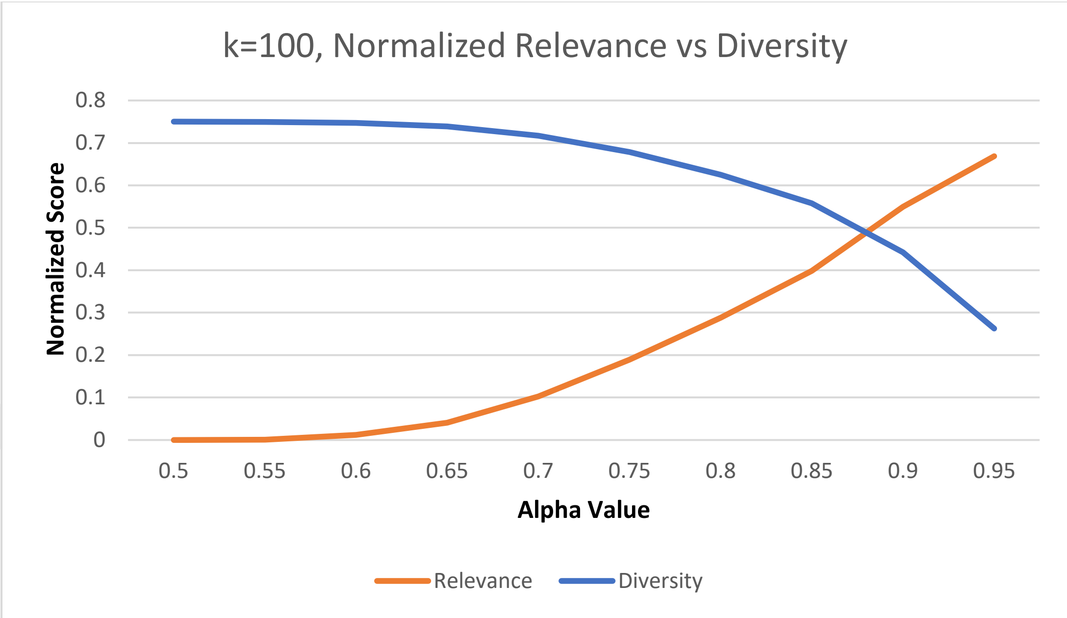

Figure 4.3: Normalized relevance and diversity scores with respect to the alpha value for k=100

In these analyses, estimated the *alpha* values using the whole experiment sets we used. In order to refine the optimal *alpha* value, we need to revisit this estimation, after we refine our parameters and do the analysis on a more optimal parameter set. At this point we can safely state that the optimal *alpha* parameter will be in range of [0.7, 0.9].

4.5. OPTIMAL PARAMETERS AND DIVERSITY ANALYSIS

The four parameters of the search algorithm, stemming, feature reduction ratio, C value and the *alpha* value are the hyper parameters we want to optimize. Up to this point, we analyzed each parameter using the whole experiment configurations. In this section, we want to fine tune each parameter further. In order to find the optimal value of a hyper-parameter, optimizing that parameter over the whole experiment space is not a perfect approach. Instead, we want to focus on a more optimal subset. With the knowledge, we gained from the first four sections; we want to limit the experiment space so our analysis can find more optimal hyper-parameters.

In Section 1, we found out that stemming works best for more cases. Although for a few experiment configurations, the diversity scores drop somewhat, for most configurations stemming proved to be the better option. In Section 2, we found out that the feature reduction ratio does not affect the performance significantly. Similar to stemming, for a few experiment configurations, a little variation in relevance and diversity scores are observed, but for most cases, we observed almost no effect. So we concluded that the feature reduction ratio of 10% is the best, since it provides best feature reduction without compromising the performance.

In Section 3, we analyzed the C parameter. Unlike previous two parameters, we could not find a clear cut best value for this parameter. In Section 4, we analyzed the *alpha* parameter. We narrowed the optimal *alpha* value to a smaller range. But we could not find pinpoint the optimal *alpha* value.

In the first 4 sections, we analyzed the hyper-parameters using all 432 experiment configurations. Since we fixed the first two hyper-parameters to using the stemming and using 10% feature reduction ratio we can eliminate 5/6 of this set. We also narrowed the *alpha* values to a smaller range, including only 5 values instead of 12. Thus, we reduced the 432 experiment configurations to only 30. We will try to find the optimal C parameter using this smaller experiment sets first. Then we will try to optimize the *alpha* value further.

Table 4.6 presents the results; we repeated with narrowed experiment sets for the number of category parameters. With the narrowed experiment set, we did not observe any certain value shine through. Rather, the differences we observed in our general experiment diminish a little. Accordingly, we believe the C value is not a very important parameter for our purpose. On the other hand, we wanted to analyze the results further by distinguishing the results by k values.

Table 4. 6*: Results of C=100 and C=200 experiments compared to the C=50 experiments*

Ratio	C=100 wrt. C=50		C=200 wrt. C=50	
Average Scores	Relevance	Diversity	Relevance	Diversity
General Experiment Set	0.9452	0.9534	0.9787	0.9840
Narrowed Experiment Set	0.9755	0.9370	0.9905	0.9787
General Experiment Set, k=10	1.0106	1.0099	1.0252	1.0252
Narrowed Experiment Set, k=10	1.0099	1.0126	1.0194	1.0311
General Experiment Set, k=100	0.8798	0.8970	0.9322	0.9427
Narrowed Experiment Set, k=100	0.9412	0.8615	0.9616	0.9262

For k=10, the narrowed experiment set gave quite similar results to general experiment set. The C value of 100 performs slightly better than 50. C=200 perform slightly better than C=100. However, the differences are small enough to be explainable by statistical advantages of higher C values. For k=100 case, the narrowed experiment result seems to be better than general experiment results for 100 and 200 categories. However, C=50 still outperform them significantly.

Table 4. 7*: Results for number of average relevance scores and average diversity score increase compared to the maximum relevance configuration, for different C and k values with narrowed experiment set.*

Narrowed Experiment Set	Average Relevance Score			Average Diversity Score Increase w.r.t. α=1.0 case		
	All Experiments	k=10	k=100	All Experiments	k=10	k=100
C=50	0.3607	0.4104	0.3111	1.6301	1.6349	1.6253
C=100	0.3537	0.4143	0.2930	1.6582	1.5307	1.7857
C=200	0.3587	0.4181	0.2993	1.5672	1.4643	1.6701

Table 4.7 displays the results for our diversity score increase with respect to *alpha*=1.0 case approach. The results are similar to our earlier analysis with generalized dataset. However, the values are milder compared to the general dataset.

For k=10, 50 categories fared the best, 100 categories the next and the 200 categories fared the worst. For k=100, 100 categories improved the best, followed by 200 categories and 50 categories fared the worst.

With the result we get, selecting a good C value is hard. The results we get are conflicting for different analysis we made. No values performed well for all the cases. However, the differences between the scores of the number of category values are small. So we think this parameter is not that important. However, any of this values or any other value might work similar or even better in practice. We selected C value of 50. We simply selected this value because we need to use a determined value. No value outperformed all others and we believe 50 would work as well as any others. We also selected this value because it is smallest, thus it will bring the smallest overhead.

For further tuning the *alpha* values, we repeat the analysis we made in Section 4 for only experiments with stemming on, feature reduction ratio of 10% and the number of clusters of 50. For the first approach to determining *alpha*, i.e. weighting from the average relevance and diversity scores, we only considered the alpha values we used in the refined experiments; 0.7 through 0.9. We observed that average relevance score for these experiments is 0.3577 while average diversity score is 0.9197. Thus the *alpha* estimate with first approach can be computed as follows;

$$0.3577\,\alpha \;= 0.9197(1-\alpha) \tag{20}$$

With this approach we find the *alpha* as 0.7200. When we repeat our second approach of *alpha* estimation with our narrowed experiment set, we get Figure 4.4 if we compute the *alpha* value by linear interpolation, we can estimate the *alpha* as; 0.9020.

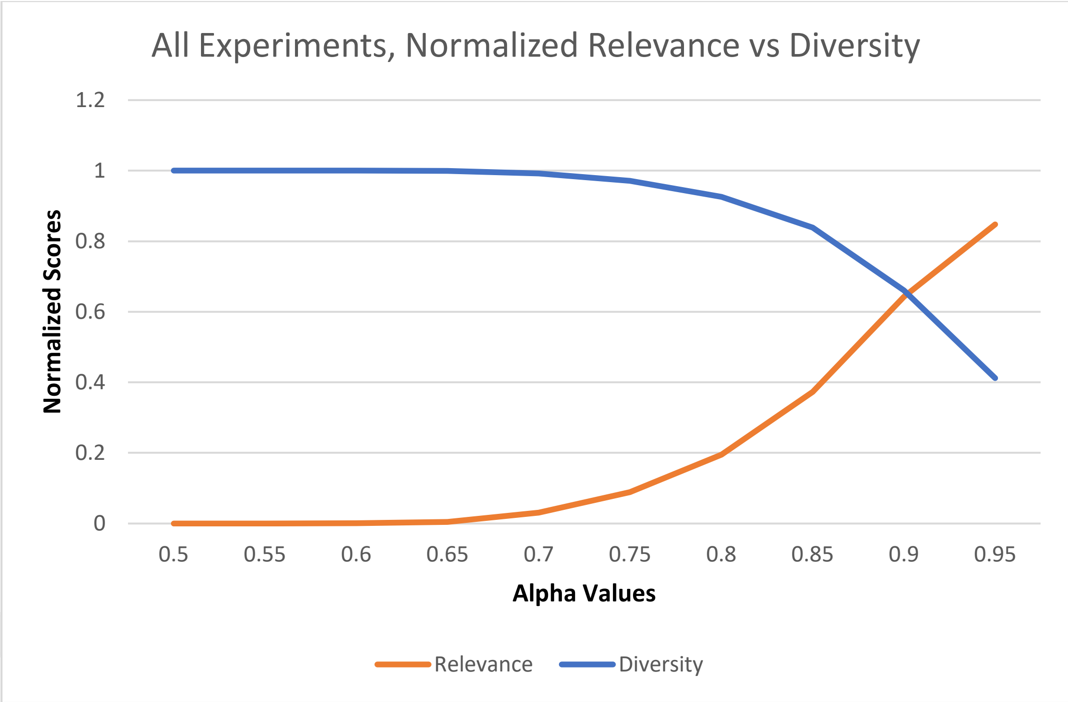

Figure 4.4: Normalized relevance and diversity scores with respect to the alpha value for narrowed experiment set

The estimated values with narrowed experiment set are pretty close to our initial estimations. Since one of our estimation inherently underestimates relevance while one inherently overestimates it, we decided to use the average of these values instead. Thus we determined the *alpha* as 0.5*(0.7200+0.9020) = 0.8110. The closest *alpha* value in our experiment set to the estimated *alpha* value is 0.8. Accordingly, we will use this value on further analysis.

After tuning the variables, we determined the best hyper-parameter values for our diversified based search algorithm is using stemming, feature reduction ratio of 10%, C value of 50 and the *alpha* value of 0.8.

4.6. EFFECTS OF RETURNED NUMBER OF RESULTS

The k parameter, the returned number of results, is a little different than other parameters. The other parameters we analyzed are the parameters for algorithmic approaches or some hyper-parameters. However, k is the search argument, like the query itself. In this section, we want to analyze how the search algorithm and our diversification approach behave for small and large k values. In our opinion, the values 10 and 100 are good values for representation of small and large user query sizes.

In general, for the same experiment configuration, both the relevance and diversity scores are a little lower. For relevance it is obvious. The average score of the top 10 results will be larger than or equal to the top 100 results. Diversity results also suffer for top 100 documents. Gathering up to 100 documents from different categories to get a high diversity score is expected to be a lot harder than gathering 10documents from different categories. These being said, the scores obtained with different k values are inherently incomparable. We need to analyze if the hyper-parameters we selected behave differently for different k values. We discussed these effects in previous sections. Thus in this section, we will simply summarize our findings.

At each experiment we done, we observed some variation. These variations are like the noise in the data and it is to be expected. Other than these variations, we only observed a discernible difference for different k values for C hyper-parameter. All other parameters behave the similarly. Of course, for some parameters, the magnitude of preference over a certain value differs for different k values. For example, stemming improves the relevance scores over non-stemming about 5.32% for k=10, while it improves 8.69% for k=100. This difference is actually a significant finding. However, since stemming is clearly beneficial for both k values, we did not have any reason to analyze further. For feature reduction ratio and *alpha* values, the experiment results were similar for different k values. However, the C value results are inherently more complicated.

For k=10 case, the number of selected documents are much smaller than the number of categories. Accordingly, in order to reach highest diversity scores, search algorithm does not need to include documents from categories quite unrelated to the query. Since selecting 10 documents of different categories from 200 categories is easier than selecting them from 50 documents, for k=10 case, we observe a small improvement as the C value increase. However, for k=100 case, particularly for 100 categories, search algorithm needs to select one document from each category to get highest diversity score. Thus search algorithm need to select a very high number of unrelated documents just because they are of an unselected category to improve its diversity scores. Basically, the number of categories becomes a limiting factor for diversification. In our experiments, for k=100, higher diversity scores are expected to be reached as the C value increase. However, since with 50 categories, we could not select 100 documents of different categories, we relaxed the diversity conditions for this case. Thus, the search algorithm was able to reach higher diversity scores much easier for 50 categories than 100 categories. 100 categories performed the worst. 200 categories improved over 100 categories but still to reach higher diversity scores search algorithm has to select a very high ratio of documents from unrelated categories, even for 200 categories. So, 50 categories seem to work well, but only because of its relaxed diversity scoring approach.

4.7. EVALUATION OF DIVERSIFIED SEARCH ALGORITHM

Diversification and relevance are somewhat opposing goals. If we want to improve diversity, the relevance has to decrease somewhat. The success of the diversified search algorithm lies in its ability to gain much diversity by sacrificing little relevance. This way, users can still get acceptably relevant documents from a much diverse set of categories.

In section 4.6, we determined the optimal hyper-parameters for our algorithm. Use of stemming and the feature reduction ratio are the decisions that affect the search algorithm, regardless of it being diversified or not. Accordingly, those parameters are more related to library data. Thus they affect the results the same way for both diversified and non-diversified searches. *Alpha* parameter is the weighing parameter between diversity and relevance. Thus, for non-diverse search algorithm, the *alpha* value has to be 1.0. And the C parameter is used only for evaluation of diversity scores. Thus it does not affect non-diverse search results.

In this section, we compared the results for four groups of experiments. First one is, our tuned parameters; stemming is on, feature reduction ratio of 0.1, 50 categories and *alpha* of 0.8. The second group of experiments will be the non-diversif The first two columns of Table 4.10 shows, diversified search configuration with tuned variables (G1), versus the same configuration except the search reaching for maximum relevance (G2).

ied configuration of the tuned experiments. The parameters of the second group are; stemming is on, feature reduction ratio of 0.1, 50 categories and *alpha* of 1.0. Third group of experiments are with the same configuration with the first two groups but with *alpha*=0.0. Thus, these experiments try to blindly maximize the diversity. And lastly, we want to compare the generic case which is not tuned for library dataset. The stemming is not usually used by default, thus for this group of experiments stemming is off. Feature reduction is not used by default. Ideally we should select feature reduction ratio of 1.0. However, that is not in our experiment set, so we selected the closest value of 0.5. That seems like a quite significant difference. However, the analysis of the library dataset shows us, with the feature reduction by half, we only eliminate the features that appear only in a single document. Thus, they do not actually matter a lot. And they do not affect the similarity scores significantly. So, our selection can give a representative result. Since the C parameter is used for diversity computations, we need to use the same C value with our other groups of experiments. Otherwise the results would not be comparable. Lastly, we used alpha=1.0, since generic algorithm would not care for diversification. Each group would contain a experiment for 2 k values we used. Thus, our overall comparison would consist of 8 experiments. Table 4.8 summarizes the experiment configurations.

Table 4. 8*: Selected experiment configurations*

Experiment Set	Stemming	Feature Reduction Ratio	C	*Alpha*	Configuration Description
G1	On	10	50	0.85	Tuned Variables, Diversified
G2	On	10	50	1.0	Tuned Variables, Non-Diversified
G3	On	10	50	0.0	Tuned Variables, Max Diversity
G4	Off	50	50	1.0	Default Variables

Table 4. 9: *Comparison of selected experiment configurations*

k	Experiment Set	Relevance Score	Diversity Score
10	G1	0.4090	0.9389
	G2	0.4462	0.5616
	G3	0.3887	1.0000
	G4	0.4241	0.5556
100	G1	0.3079	0.9967
	G2	0.3339	0.6011
	G3	0.3069	1.0000
	G4	0.3082	0.6046

Table 4.9 shows the results for the experiment configurations we are comparing. G1 results, which are our tuned parameters with diversification, are as expected gave a little worse relevance score than G2 configuration. G2 configuration ultimately tries to maximize the relevance. The diversity of G1 on the other hand is much better than G2. The G3 experiments, which blindly tries to maximize diversity performs slightly better than G1 in diversity scores, but is much worse in terms of relevance. G4 experiments, which are the default parameters performed slightly better than G1 for relevance, since it also tries to maximize for relevance. But it performs much worse in diversity. In addition, we observe that G2 experiments perform better than G4, default parameter experiments, for relevance. And they perform almost identical for diversity scores. This shows that our stemming and feature reduction ratio parameters are well suited for library dataset.

Table 4. 10: *Relative comparison of relevance and diversity scores between diversified searches with tuned variables versus other configurations*

	G1 vs G2		G1 vs G3		G1 vs G4	
	Relevance	Diversity	Relevance	Diversity	Relevance	Diversity
k = 10	0.9167	1.6718	1.0521	0.9389	0.9645	1.6900
k = 100	0.9219	1.6580	1.0032	0.9967	0.9990	1.6485
Average	0.9194	1.6649	1.0277	0.9678	0.9817	1.6692

As expected, our approach performed a little worse, about 8%, in terms of relevance. However, in terms of diversity, this 8% drop allows the algorithm to increase its diversity score for about 66%. We think this is a good tradeoff. The comparison for default parameters (G4), is given in last two columns. The default parameters experiments, also aim to maximize relevance similar to G2. We observe a comparison between G1 and G4 to the comparison between G1 and G2. But the tradeoff is even better for G1. For k=10, we get about 3.5% drop in relevance while for k=100, the relevancies are almost identical. On the other hand, for k=10, we get 69% increase in diversity score, while for k=100 the increase is 65%. These values show the improvements of our tuned configuration over the default case.

For k=10, the maximum diversity configuration (G3) performs worse than our diversified search configuration with tuned parameters (G1) in terms of relevance but performs better in terms of diversity. The ratios in Table 4.10 may look like G3 more than makes up for its relevance loss in its diversity increase. But when we compare these results deeper, we can see from Table 4.9, although the diversity score for G3 is full, the diversity score for G1 is very close to the top. This means the results for G1 are sufficiently diverse already and the increase in diversity by G3 is not as important as the score implies. Thus, we believe the relevance advantage of G1 over G3 is more important. For k=100, G1 and G3 performed quite close.

In summary, we can say that the use of stemming and the feature reduction ratio of 0.1 definitely work well for the library dataset. We can see the obvious advantage, when we compare G2, tuned parameter non-diversified configuration with the G4, the default configuration. On top of that, we can see the effects of diversified tuned parameter configuration when we compare G1 and G2. G1 loses some relevance score, but more than makes up for it in diversity scores. Accordingly, we believe, the tuned parameters we found and the diversified search algorithm works well for the library dataset and we believe it can work fine in the real world. We might need to tune the C and the *alpha* value depending on the user experiences, but in general our approach would work well.

4.8. ANNOTATED EXPERIMENTS

We have performed many experiments comparing our approach to currently implemented search algorithm. For comparison, we used the relevance and diversity scores we defined. Our relevance score is based on cosine similarities of queries and documents. Diversity score depends on how many different categories represented in the result set. However the category assignments are done using the methods discussed in the classification sections. We tried to define our metrics so they will reflect the real user experience as best as possible. However, we cannot objectively state that this is the case.

In order to improve our work in this respect, we performed an annotated experiment with smaller scale. We decided to sample 100 queries from the queries dataset. Then run these queries with using our search algorithm and actual library search system (OPAC). We get top 10 results and manually compared our results to OPAC's results. The scale of this experiment is relatively small and it may not be generalized. However, the results of this experiment can be used as an indicator of real world performance of our algorithm.

We selected the 100 queries we used in this analysis in a in a random fashion while disregarding queries that contain typos. Note that eliminated queries containing typos do not introduce a bias towards any framework, as both our approach hand OPAC do not provide facilities to handle typos.

After we selected 100 queries for annotated experiments, we run these queries on both our search system and on OPAC search system. We manually analyzed both search results for each query. For relevance, we considered a search result either relevant or irrelevant. We used this binary approach for its simplicity. For a result set, we count the number of relevant documents in the result set and give the relevance score according to the ratio of relevant documents over the whole result set. For diversity score, we manually categorize the documents. Then we count the number of different categories represented in the result set. We computed the diversity score with the same formulation discussed in Section 3.3.2.

Table 4.11 and Table 4.12 show two example query results for both our optimal configuration and for OPAC system. For the query "turizm", our approach works quite well. We managed to improve diversity quite significantly without losing relevance. OPAC system performed very poorly in terms of diversity. Its results do not only belong to the same category, but they are essentially different issues of the same periodical publication.

Table 4. 11: Annotated comparison of G1 and OPAC search results for the query "turizm"

G1 Results	**Rel**	**Cat**	**OPAC Results**	**Rel**	**Cat**
turizm istatistikleri	✓	A	Van turizm envanteri	✓	A
turizm	✓	B	1981 (Ocak ayı) Turizm Hareketleri	✓	B
özbekistan turizm rehberi	✓	C	1980 Turizm hareketleri	✓	B
turizm mevzuatı	✓	D	Ordu ili kültür ve turizm envanteri	✓	A
kırsal turizm	✓	E	1980 (6 aylık) turizm hareketleri	✓	B
termal turizmi	✓	F	1981 (5 aylık) turizm hareketleri	✓	B
kış turizmi	✓	G	Aksaray turizm envanteri	✓	A
turizm rehberi	✓	G	Turizm hakkında raporlar	✓	B
turizm ii	✓	I	1980 (4 aylık) Turizm Hareketleri	✓	B
turizm eğitimi	✓	J	1980 (5 aylık) Turizm Hareketleri	✓	B
Relevance Score	1.0000		Relevance Score	1.0000	
Diversity Score	0.8889		Diversity Score	0.2222	

G1 Results	**Rel**	**Cat**
göreme	×	A
göreme	×	A
görme engelliler hakkında görenler için rehber	✓	B
görme engelliler hakkında görenler için rehber	✓	B
beceri gelişimine yönelik oyunların görme engellilere uyarlaması serimer zuhal	✓	C
görme engelli kadınlar kurultayı 28 29 haziran 2007	✓	C
görme engellilere yönelik sağlık rehberi	✓	D
görme engelli üniversite öğrencilerinin bilgiye erişim sorunları aydın emet aynur	✓	E
ekpss görme engelli adaylar için	✓	F
işitme engelli öğrencilerde binoküler görme keleş sadullah	✓	G
Relevance Score	0.8000	
Diversity Score	0.6667	

For the query "görme engelliler", our approach improves the diversity significantly. We managed to give results from seven different categories, while OPAC system gave results from only three categories. In addition, five of ten results of the OPAC system are the multiple instances of the same publication. For relevance, our approach failed to match OPAC system. Two of the results are not relevant to the query. Accordingly, our relevance is worse for this query but diversity improvement is much higher.

***Table 4. 12**: Annotated comparison of G1 and OPAC search results for the query "görme engelliler"*

OPAC Results	**Rel**	**Cat**
Formation des aveugles des campagnes dans les régions économiquement peu développées	✓	A
Görme engelliler ilköğretim okulları matematik dersi öğretim programı	✓	A
Görme engelliler hakkında görenler için rehber	✓	B
Görme engelliler hakkında görenler için rehber	✓	B
Görme engelliler hakkında görenler için rehber	✓	B
Okul öncesi dönemdeki az gören çocuğun izleme becerilerinin gelişiminde işlevsel görme aktivite programının (İGAP izleme) etkisi: bir örnek olay çalışması / Aslan, Cem	✓	C
The art and science of audio book production	✓	B
Görme engelliler ilköğretim okulları matematik dersi öğretim programı	✓	B
Görme engelliler hakkında görenler için rehber	✓	B
Görme engelliler hakkında görenler için rehber	✓	B
Relevance Score	1.0000	
Diversity Score	0.2222	

The reason for irrelevant results is the stemming approach we used in our search approach. The stem of the word "görme" is "gör". Although the irrelevant results; "göreme" refers to a place, it has a second meaning whose stem is also "gör". This example shows the stemming approach leading to such irrelevant results. Although in our analysis, the stemming approach turned out to be quite beneficial, in reality it might lead to irrelevant results like this. Thus in practice stemming might not be as beneficial as the analysis show.

***Table 4. 13:** Annotated comparison of G1 and OPAC search results for average relevance and diversity*

	G1 Results	OPAC Results	G1 over OPAC Improvement
Average Relevance	0.8604	0.9188	-6.3561%
Average Diversity	0.6799	0.4554	49.2973%

With the annotated experiments, our framework improves the diversity 49.2973% on average. On the other hand, relevance losses are 6.3561% on average. These results are not as good as the results we obtained with our previous experiments. However, they are still significant.

Since annotated experiments require manual analysis of search results, we were only able to work on a small subset of queries. These results might not fully represent the overall user experience, but they still work as a good indicator. Also note that as, the results are not as good as our automated experiments while they indicate that the presented diversification technique may have merit.

CHAPTER 5

CONCLUSION AND FUTURE WORK

Diversification of the search results is an important concept that improves the user experience significantly. When searching a certain query, users expect to get as relevant results as possible. However, they also expect the search results to cover every aspect of the query as well. These two goals do not always coincide. Diversification based search algorithm help to balance the relevance and the diversity of the search results to give the user best experience possible.

The relevance and diversity are quite context specific concepts. Different applications require different approaches to quantify and balance relevance and diversity. In this work, we used a real-world dataset obtained from the National Library of Turkey. This dataset contain all the items presented in the library catalogue and all the real user queries. We tried to apply and optimize the diversified search approach to this dataset. Since we work on a real-world dataset, our results are a good representation of the real user experiences.

In this work, we implemented and modified a diversified search algorithm similar to MMR based search algorithms. Our search algorithm is inspired from the swap approaches and local search approaches frequently used in web search systems. We tuned the hyper-parameters and algorithmic approaches so that they fit the dataset best. In the end we improved the diversity scores while losing relatively little relevance.

We started with formatting the dataset. This step is important in order to be able to compare related documents in a normalized way. Then we analyzed if we should use stemming approach. Stemming approach require to use the stem of the word instead of the word itself. This allows to better relate the documents containing words with the same stem. However, we also lose information coming from suffixes. After experimenting, we found out that stemming works well for the library dataset and improve the relevance while not affecting diversity.

Then we analyzed the feature selection strategy we should use. We used the info-gain algorithm to determine the relative information contents of each feature. Then we experimented three feature reduction ratios. We found out that we could reduce the number of features in the dataset to 1/10of initial size without affecting the relevance and diversity scores. Thus, we decided to use 0.1 feature reduction ratio.

In order to quantify the diversity, we needed to categorize the documents. In order to do that, we used LDA algorithm to classify the documents according to their similarities. We used several number of categories to experiment. However, we did not find any number of categories better than others. In some situations, one performed better while in some situations other performed better. In general, all the number of categories we experimented performed reasonably well and we selected to use the one with the least overhead. We believe, the optimal number of categories depends on the user preferences and we cannot find it via analysis of the dataset. This depend on what does the user considers as diverse. Due to the nature of categorization, each topic belongs to a subcategory branching of the categorization tree. Accordingly, each topic can be considered as a part of many categories; some a quite specific and some are more general, on the category tree. If the user prefer diversification within a more specific area, than having higher number of categories would fit the user better. On the other hand, if the user wants the diversification as general as possible, then the lower number of categories fit better. In order to find the optimal value for the number of categories, we need to obtain and analyze the user feedback.

Balancing between relevance and diversity require a weighting approach that combines the relevance and diversity scores. We designed and computed the relevance and diversity scores, so that the values would be normalized. Thus, we were able to use a linear weighting approach to combine relevance and diversity scores. The optimal weighting between relevance and diversity depends on the user preference. However, in this work, we tried to find an optimal value that gives about equal importance to both of these scores.

After tuning all the variables, we were able to improve the diversity while losing a little relevance. The detailed analysis is given in Chapter 4.7. Compared to default search algorithm, on average our tuned parameter diversified search algorithm were able to improve diversity score by 66.7% while losing the relevance by only 1.9%. This is a quite significant improvement and we believe this can improve user experience significantly. We believe if we are able to apply our work at the library search system, we would be able to give a much better user experience through providing much diverse but still quite relevant search results.

5.1. FUTURE WORK

We showed that much diverse results can be achieved with our algorithm without losing too much relevance. However, there is still a lot of room for improvement. Our current work serves as a proof of concept. It can still work well as is in the real world. But we can improve it further. The main area of improvement is about the quantification of diversification.

In this work, we assign a certain category to each document. Then we tried to maximize the total number of categories represented within a search result set. However, this approach does not represent the real-world categorization quite accurately. A document does not belong to a single category. Instead it can belong to multiple categories. Each of these categories can have multiple weights. We can incorporate this characteristic into our diversity quantification approach. Instead assuming each document represent a certain category, we can consider each document representing multiple categories, with certain weights. This way we can represent the real world categorization better. Implementation of this approach is also compatible with our categorization technique. The LDA method we use, does not simply assign a category to a document. But rather, it gives a list of probabilities that this document belonging to each category. We can use this information to build the list of categories that this document belongs to.

Another improvement we can make about diversification is about tuning the number of categories parameter. As mentioned earlier, a document does not only belong to a certain category. But it belongs to a branch of categories form the categorization tree. Thus, when we classify the documents into a lower number of categories, we assign them into more generalized parts of these branches. When we classify the documents into high number of categories, we assign them into more specified categories. Instead of tuning the number of categories parameter, we can augment them into a single multi category feature.

If we classify the documents into a certain number of categories using LDA algorithm, we get the weighted categorization of each document into these categories. If we repeat this classification algorithm using several number of categories, we can get category assignments of the same document with different category specificities. If we combined these assignments together, we can get category assignments that cover general, specific and in-between categories. If we use this category feature while computing the diversity scores, we can represent the actual diversification much better than our current approach. We plan to implement this approach as a future work.

REFERENCES

Agrawal, R., Gollapudi, S., Halverson, A., & Ieong, S. (2009). Diversifying Search Results. *WSDM'09: Second ACM International Conference on Web Search and Web Data Mining*, (s. 5-14).

Akın, A. A. (2019, July 23). *Zemberek-NLP provides Natural Language Processing tools for Turkish.* Retrieved February 2, 2021, from githup: https://github.com/ahmetaa/zemberek-nlp

Bagirov, A., Rubinov, A., & Yearwood, J. (2002). A Heuristic Algorithm for Feature Selection Based on Optimization Techniques. In *Heuristic and Optimization for Knowledge Discovery* (pp. 13-26).

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research* , 993-1022.

Brownlee, J. (2020, August 15). *An Introduction to Feature Selection*. Retrieved March 20, 2021, from Machine Learning Mastery: https://machinelearningmastery.com/an-introduction-to-feature-selection/

Cambazoglu, B. B., Kayaaslan, E., Jonassen, S., & Aykanat, C. (2013). A Term-Based Inverted Index Partitioning Model for Efficient Distributed Query Processing. *ACM Transactions on the Web* , Vol. 7, No. 3, Article 15.

Carbonelland, J. G., & Goldstein, J. (1998). Theuse of mmr, diversity-based reranking for reordering document sand producing summaries 335–336, 1998. *SIGIR , 98*, 335-336.

Chandrashekar, G., & Şahin, F. (2014). A Survey on Feature Selection Methods. *Computers and Electrical Engineering , 40* (1), 16-28.

Clipa, T., & Di Nunzio, G. (2020). A Study on Ranking Fusion Approaches for the Retrieval of Medical Publications. *Information* , 103.

Gollapudi, S., & Sharma, S. (2009). An axiomatic approach for result diversification. *WWW '09: Proceedings of the 18th International Conference on World Wide Web*, (s. 381-390).

Göynük, B. (2019). *Supervised Learning Methods for Image Search Result Diversification.* Master Thesis, Middle East Technical University, Ankara.

Gültekin, V., Kılıç, N., Balta, S., Özdemir, F. S., & Karaağaç, D. (2016). *Ankara Üniversitesi Kütüphaneleri Kataloglama - Marc Uygulamaları ve Örnekleri.* Ankara.

He, J., Meij, E., & Rijke, M. d. (2011). Result Diversification Based on Query-Specific Cluster Ranking. *J. Assoc. Inf. Sci. Technol. 62(3)* , 550-571.

Jiangand, J., & Lee, S. (2007). A Weight-based Feature Extraction Approach for Text Classification. *Second International Conference on Innovative Computing, Informatio and Control (ICICIC 2007),* (s. 164-164).

Keskinkilic, M. (2011). Bilgi Otomasyonu ve Bir Kütüphane Bilgi Sistemi Ana Modülü Üzerine Prototip Uygulamasi. *Bartin Univ. I.I.B.F. Dergisi* .

Keswani, V., & Celis, L. E. (2020). Implicit Diversity in Image Summarization. *Proceedings of the ACM on Human-Computer Interaction* , 1-28.

Kowsari, K., Jafari Meimandi, K., Heidarysata, M., Mendu, S., Barnes, L., & Brown, D. (2019). Text Classification Algorithms: A Survey. *Information , 10* (4), 150.

Ladha, L., & Deepa, T. (2011). Feature Selection Methods and Algorithms. *International Journal on Computer Science and Engineering* , 1787-1797.

Law, M. H., Figueiredoand, M. A., & Jain, A. K. (2004). Simultaneous Feature Selection and Clustering Using Mixture Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence , 26* (9), 1154-1166.

Leuken, R. H., Garcia, L., Olivares, X., & Zwol, R. v. (2009). *Proceedings of the 18th international conference on World wide web.*

Luhaniwal, V. R. (2020, October 24). *A comprehensive guide to Feature Selection using Wrapper methods in Python.* Retrieved June 14, 2021, from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2020/10/a-comprehensive-guide-to-feature-selection-using-wrapper-methods-in-python/

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval.* Cambridge University Press.

McDonald, G. (2009). Ridge regression. *Wiley Interdisciplinary Reviews: Computational Statistics* , 93 - 100.

Naini, K. D., Altingovde, I. S., & Siberski, W. (2016). Scalable and Efficient Web Search Result Diversification. *ACM Transactions on the Web 10(3)* , 15:1-15:30.

Özdemiray, A., & Altıngövde, I. (2013). *Score and Rank Aggregation Methods for Explicit Search Result Diversification.* Technical Report METU-CENG-2013-01, Middle East Technical University, Ankara.

Paul, S. (2020, January 2). *Beginner's Guide to Feature Selection in Python*. Retrieved 02 15, 2021, from datacamp: https://www.datacamp.com/community/tutorials/feature-selection-python

Pearson, K. (1900). On The Criterion That A Given System Of Deviations From The Probable In The Case Of A Correlated System Of Variables Is Such That It Can Be Reasonably Supposed To Have Arisen From Random Sampling. *Philosophical Magazine* , 157-175.

Pedregosa, F. a. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* , 2825-2830.

Pritchard, J. K., Stephens, M., & Donnelly, P. (2000). Inference of population structure using multilocus genotype data. *Genetics* , 945-959.

Qaiser, S., & Ali, R. (2018). Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents. *International Journal of Computer Applications* , 25-29.

R., S. A. (2012). *Multivariate filter with particle swarm optimisation variants for feature selection.* Selangor: Universiti Kebangsaan Malaysia.

Revert, F. (2018, December 17). *An overview of topics extraction in Python with LDA.* Retrieved February 2, 2021, from Towards Data Science: https://towardsdatascience.com/the-complete-guide-for-topics-extraction-in-python-a6aaa6cedbbc

Rohini, R. M. (2016). LASSO: A feature selection technique in predictive modeling for machine learning. *IEEE International Conference on Advances in Computer Applications* , 18-20.

Santos, R., Peng, J., Macdonald, C., & Qunis, I. (2010). Explicit search result Diversification through Sub-queries. *ECIR*, *5993*, s. 87-99.

Sarikaya, S. (n.d.). *Kasif: Milli Kutuphane*. Retrieved from Milli Kutuphane: https://kasif.mkutup.gov.tr/

Sarkar, D. (2018, January 30). *Traditional Methods for Text Data.* Retrieved February 2, 2021, from Towards Data Science: https://towardsdatascience.com/understanding-feature-engineering-part-3-traditional-methods-for-text-data-f6f7d70acd41

Saurabh Paul, P. D. (2015). Feature Selection for Ridge Regression with Provable Guarantees. *Neural computation* .

Tina. (2020, January 3). *Feature Engineering Methods for Text Data.* Retrieved February 16, 2021, from medium.com: ¬ https://medium.com/@tinakindenproperty/feature-engineering-methods-for-text-data-fc7e96a886ab

Uğuz, H. (2011). A two-stage feature selection method for text categorization by using information gain, principal component analysis and genetic algorithm. *Knowledge-Based Systems* , 1024-1032.

Ulu, Y. B. (2019). *Limitations and Improvement Opportunities for Implicit Result Diversification in Search Engines.* Master Thesis, TED University, Ankara.

Vieira, M. R., Razente, H. L., Barioni, M. C., Hadjieleftheriou, M., Srivastava, D., Traina Jr., C., et al. (2011). On query result diversification. *ICDE* , 1163-1174.

Visalakshi, S., & Radha, V. (2014). A Literature Review of Feature Selection Techniques and Applications: Review of Feature Selection in Data Mining. *2014 IEEE International Conference on Computational Intelligence and Computing Research*, (s. 1-6).

Wah, Y., Ibrahim, N., Hamid, H., Rahman, S., & Fong, S. (2018). Feature Selection Methods: Case of Filter and Wrapper Approaches for Maximising Classification Accuracy. *Pertanika Journal of Science & Technology* , 329-340.

Wu, S., Zhang, Z., & Xu,, C. (2019). Evaluating the Effectiveness of Web Search Engines on Results Diversification. *Information Research: An International Electronic Journal , 24* (1).

Yang, F., Shang, F., Huang, Y., Cheng, J., Li, J., Zhao, Y., et al. (2017). Lftf: A frame work for efficient tensor analytics at scale. *Proceedings of the VLDB Endowment , 10* (7), 745-756.

Yang, Y., & Pedersen, J. (1997). A Comparative Study on Feature Selection in Text Categorization. *Proceedings of the Fourteenth International Conference on Machine Learning*, (s. 412-420).

Yigit-Sert, S., Altingovde, I. S., Macdonald, C., Ounis, I., & Ulusoy, Ö. (2021). Explicit Diversification of Search Results Across Multiple Dimensions for Educational Search. *Journal of the Association for Information Science and Technology* , 315-330.

Zarepoor, M., & Seeja, K. (2015). Feature Extraction or Feature Selection for Text Classification: A Case Study on Phishing Email Detection. *International Journal of Information Engineeringand Electronic Business , 7* (2), 60-65.

Zou, H. a. (2005). Regularization and Variable Selection via the Elastic Net. *Journal of the Royal Statistical* , 301-320.

Zuccon, G., Azzopardi, L., Zhang, D., & Wang, J. (2012). Top-k Retrieval Using Facility Location Analysis. *ECIR* , 305-316.

APPENDICES

Appendix 1 – Experiment Results

k	Stemming	Feature Reduction Ratio	Category Count	Alpha	Relevance Score	Diversity Score
10	0	10	50	0	0.365509	1.000000
10	0	10	50	0.5	0.365509	1.000000
10	0	10	50	0.55	0.365518	0.999989
10	0	10	50	0.6	0.365549	0.999945
10	0	10	50	0.65	0.366300	0.998680
10	0	10	50	0.7	0.370134	0.990529
10	0	10	50	0.75	0.377680	0.970317
10	0	10	50	0.8	0.388583	0.931718
10	0	10	50	0.85	0.399144	0.880610
10	0	10	50	0.9	0.410421	0.799109
10	0	10	50	0.95	0.419870	0.679483
10	0	10	50	1	0.424065	0.567649
10	0	10	100	0	0.372255	1.000000
10	0	10	100	0.5	0.372255	1.000000
10	0	10	100	0.55	0.372266	0.999989
10	0	10	100	0.6	0.372290	0.999955
10	0	10	100	0.65	0.372848	0.999009
10	0	10	100	0.7	0.375991	0.992312
10	0	10	100	0.75	0.382463	0.975017
10	0	10	100	0.8	0.391852	0.941777
10	0	10	100	0.85	0.401300	0.896000
10	0	10	100	0.9	0.411579	0.821537
10	0	10	100	0.95	0.420274	0.711041
10	0	10	100	1	0.424065	0.608471
10	0	10	200	0	0.379768	1.000000
10	0	10	200	0.5	0.379768	1.000000
10	0	10	200	0.55	0.379768	1.000000
10	0	10	200	0.6	0.379783	0.999978
10	0	10	200	0.65	0.380135	0.999375
10	0	10	200	0.7	0.382278	0.994810
10	0	10	200	0.75	0.387005	0.982171
10	0	10	200	0.8	0.394458	0.955744
10	0	10	200	0.85	0.402688	0.915758
10	0	10	200	0.9	0.411993	0.848058
10	0	10	200	0.95	0.420333	0.741468
10	0	10	200	1	0.424065	0.640741
10	0	10	300	0	0.358972	1.000000
10	0	10	300	0.5	0.358972	1.000000
10	0	10	300	0.55	0.358983	0.999987
10	0	10	300	0.6	0.359001	0.999963
10	0	10	300	0.65	0.359616	0.998924
10	0	10	300	0.7	0.363048	0.991603
10	0	10	300	0.75	0.369387	0.974648
10	0	10	300	0.8	0.379675	0.938056
10	0	10	300	0.85	0.390342	0.886399
10	0	10	300	0.9	0.404377	0.784970
10	0	10	300	0.95	0.417839	0.615623
10	0	10	300	1	0.424065	0.454199
10	0	30	50	0	0.366699	1.000000
10	0	30	50	0.5	0.366699	1.000000

10	0	30	50	0.55	0.366710	0.999987
10	0	30	50	0.6	0.366737	0.999949
10	0	30	50	0.65	0.367507	0.998646
10	0	30	50	0.7	0.371209	0.990780
10	0	30	50	0.75	0.378635	0.970939
10	0	30	50	0.8	0.389129	0.933805
10	0	30	50	0.85	0.399328	0.884447
10	0	30	50	0.9	0.410557	0.803401
10	0	30	50	0.95	0.419908	0.684841
10	0	30	50	1	0.424065	0.573739
10	0	30	100	0	0.370851	1.000000
10	0	30	100	0.5	0.370851	1.000000
10	0	30	100	0.55	0.370862	0.999989
10	0	30	100	0.6	0.370882	0.999960
10	0	30	100	0.65	0.371501	0.998914
10	0	30	100	0.7	0.374863	0.991751
10	0	30	100	0.75	0.381602	0.973725
10	0	30	100	0.8	0.391379	0.939096
10	0	30	100	0.85	0.400960	0.892725
10	0	30	100	0.9	0.411481	0.816470
10	0	30	100	0.95	0.420123	0.706582
10	0	30	100	1	0.424065	0.600827
10	0	30	200	0	0.379608	1.000000
10	0	30	200	0.5	0.379608	1.000000
10	0	30	200	0.55	0.379609	0.999999
10	0	30	200	0.6	0.379623	0.999979
10	0	30	200	0.65	0.379974	0.999383
10	0	30	200	0.7	0.382161	0.994730
10	0	30	200	0.75	0.386832	0.982230
10	0	30	200	0.8	0.394416	0.955409
10	0	30	200	0.85	0.402643	0.915379
10	0	30	200	0.9	0.412041	0.847018
10	0	30	200	0.95	0.420292	0.741747
10	0	30	200	1	0.424065	0.640017
10	0	30	300	0	0.360916	1.000000
10	0	30	300	0.5	0.360916	1.000000
10	0	30	300	0.55	0.360927	0.999988
10	0	30	300	0.6	0.360941	0.999969
10	0	30	300	0.65	0.361498	0.999022
10	0	30	300	0.7	0.364755	0.992071
10	0	30	300	0.75	0.370822	0.975849
10	0	30	300	0.8	0.380915	0.939992
10	0	30	300	0.85	0.391183	0.890251
10	0	30	300	0.9	0.404711	0.792403
10	0	30	300	0.95	0.417925	0.625938
10	0	30	300	1	0.424065	0.466588
10	0	50	50	0	0.363832	1.000000
10	0	50	50	0.5	0.363832	1.000000
10	0	50	50	0.55	0.363837	0.999995
10	0	50	50	0.6	0.363867	0.999952
10	0	50	50	0.65	0.364648	0.998631
10	0	50	50	0.7	0.368615	0.990202
10	0	50	50	0.75	0.376249	0.969789
10	0	50	50	0.8	0.387515	0.929943
10	0	50	50	0.85	0.398347	0.877572
10	0	50	50	0.9	0.409941	0.793976
10	0	50	50	0.95	0.419808	0.669176
10	0	50	50	1	0.424065	0.555582
10	0	50	100	0	0.368939	1.000000
10	0	50	100	0.5	0.368939	1.000000
10	0	50	100	0.55	0.368950	0.999988
10	0	50	100	0.6	0.368979	0.999947
10	0	50	100	0.65	0.369679	0.998762

10	0	50	100	0.7	0.373168	0.991338
10	0	50	100	0.75	0.379977	0.973128
10	0	50	100	0.8	0.390190	0.937019
10	0	50	100	0.85	0.400259	0.888148
10	0	50	100	0.9	0.411096	0.809758
10	0	50	100	0.95	0.420072	0.695907
10	0	50	100	1	0.424065	0.588315
10	0	50	200	0	0.379057	1.000000
10	0	50	200	0.5	0.379057	1.000000
10	0	50	200	0.55	0.379059	0.999997
10	0	50	200	0.6	0.379071	0.999980
10	0	50	200	0.65	0.379473	0.999295
10	0	50	200	0.7	0.381658	0.994633
10	0	50	200	0.75	0.386520	0.981660
10	0	50	200	0.8	0.393934	0.955387
10	0	50	200	0.85	0.402208	0.915126
10	0	50	200	0.9	0.411837	0.845175
10	0	50	200	0.95	0.420242	0.737869
10	0	50	200	1	0.424065	0.634565
10	0	50	300	0	0.362909	1.000000
10	0	50	300	0.5	0.362909	1.000000
10	0	50	300	0.55	0.362918	0.999990
10	0	50	300	0.6	0.362932	0.999971
10	0	50	300	0.65	0.363443	0.999102
10	0	50	300	0.7	0.366263	0.993079
10	0	50	300	0.75	0.371896	0.978003
10	0	50	300	0.8	0.381230	0.944825
10	0	50	300	0.85	0.391360	0.895724
10	0	50	300	0.9	0.404623	0.799632
10	0	50	300	0.95	0.417606	0.635764
10	0	50	300	1	0.424065	0.470098
10	1	10	50	0	0.388729	1.000000
10	1	10	50	0.5	0.388729	1.000000
10	1	10	50	0.55	0.388742	0.999986
10	1	10	50	0.6	0.388761	0.999958
10	1	10	50	0.65	0.389210	0.999205
10	1	10	50	0.7	0.392214	0.992799
10	1	10	50	0.75	0.398719	0.975413
10	1	10	50	0.8	0.409001	0.938922
10	1	10	50	0.85	0.419836	0.886309
10	1	10	50	0.9	0.432036	0.797481
10	1	10	50	0.95	0.442105	0.669318
10	1	10	50	1	0.446189	0.561607
10	1	10	100	0	0.395718	1.000000
10	1	10	100	0.5	0.395718	1.000000
10	1	10	100	0.55	0.395729	0.999989
10	1	10	100	0.6	0.395743	0.999968
10	1	10	100	0.65	0.396123	0.999327
10	1	10	100	0.7	0.398562	0.994117
10	1	10	100	0.75	0.404177	0.979107
10	1	10	100	0.8	0.413080	0.947522
10	1	10	100	0.85	0.422557	0.901445
10	1	10	100	0.9	0.433343	0.822600
10	1	10	100	0.95	0.442539	0.704920
10	1	10	100	1	0.446189	0.606865
10	1	10	200	0	0.404328	1.000000
10	1	10	200	0.5	0.404328	1.000000
10	1	10	200	0.55	0.404331	0.999997
10	1	10	200	0.6	0.404337	0.999988
10	1	10	200	0.65	0.404523	0.999674
10	1	10	200	0.7	0.406107	0.996294
10	1	10	200	0.75	0.409976	0.985943
10	1	10	200	0.8	0.416611	0.962374

10	1	10	200	0.85	0.424241	0.925109
10	1	10	200	0.9	0.433772	0.855158
10	1	10	200	0.95	0.442632	0.741158
10	1	10	200	1	0.446189	0.645331
10	1	10	300	0	0.373136	1.000000
10	1	10	300	0.5	0.373136	1.000000
10	1	10	300	0.55	0.373148	0.999987
10	1	10	300	0.6	0.373155	0.999976
10	1	10	300	0.65	0.373593	0.999236
10	1	10	300	0.7	0.376516	0.992979
10	1	10	300	0.75	0.382734	0.976325
10	1	10	300	0.8	0.394068	0.936132
10	1	10	300	0.85	0.406712	0.874787
10	1	10	300	0.9	0.424195	0.747905
10	1	10	300	0.95	0.440093	0.550682
10	1	10	300	1	0.446189	0.394515
10	1	30	50	0	0.387433	1.000000
10	1	30	50	0.5	0.387433	1.000000
10	1	30	50	0.55	0.387445	0.999986
10	1	30	50	0.6	0.387463	0.999962
10	1	30	50	0.65	0.387994	0.999073
10	1	30	50	0.7	0.391147	0.992361
10	1	30	50	0.75	0.397967	0.974116
10	1	30	50	0.8	0.408500	0.936802
10	1	30	50	0.85	0.419381	0.884017
10	1	30	50	0.9	0.431851	0.793222
10	1	30	50	0.95	0.442106	0.662529
10	1	30	50	1	0.446189	0.554392
10	1	30	100	0	0.394340	1.000000
10	1	30	100	0.5	0.394340	1.000000
10	1	30	100	0.55	0.394351	0.999988
10	1	30	100	0.6	0.394367	0.999965
10	1	30	100	0.65	0.394767	0.999291
10	1	30	100	0.7	0.397313	0.993850
10	1	30	100	0.75	0.403152	0.978242
10	1	30	100	0.8	0.412396	0.945446
10	1	30	100	0.85	0.422174	0.897852
10	1	30	100	0.9	0.433267	0.816841
10	1	30	100	0.95	0.442556	0.698125
10	1	30	100	1	0.446189	0.600098
10	1	30	200	0	0.403642	1.000000
10	1	30	200	0.5	0.403642	1.000000
10	1	30	200	0.55	0.403642	1.000000
10	1	30	200	0.6	0.403654	0.999983
10	1	30	200	0.65	0.403876	0.999604
10	1	30	200	0.7	0.405484	0.996166
10	1	30	200	0.75	0.409314	0.985917
10	1	30	200	0.8	0.416021	0.962093
10	1	30	200	0.85	0.423859	0.923763
10	1	30	200	0.9	0.433687	0.851539
10	1	30	200	0.95	0.442637	0.736518
10	1	30	200	1	0.446189	0.640710
10	1	30	300	0	0.382464	1.000000
10	1	30	300	0.5	0.382464	1.000000
10	1	30	300	0.55	0.382474	0.999989
10	1	30	300	0.6	0.382483	0.999976
10	1	30	300	0.65	0.382835	0.999381
10	1	30	300	0.7	0.385234	0.994236
10	1	30	300	0.75	0.390358	0.980504
10	1	30	300	0.8	0.399721	0.947223
10	1	30	300	0.85	0.410686	0.893970
10	1	30	300	0.9	0.425779	0.783946
10	1	30	300	0.95	0.440308	0.600781

10	1	30	300	1	0.446189	0.449619
10	1	50	50	0	0.387202	1.000000
10	1	50	50	0.5	0.387202	1.000000
10	1	50	50	0.55	0.387213	0.999988
10	1	50	50	0.6	0.387242	0.999947
10	1	50	50	0.65	0.387772	0.999057
10	1	50	50	0.7	0.390970	0.992264
10	1	50	50	0.75	0.397681	0.974321
10	1	50	50	0.8	0.408238	0.936950
10	1	50	50	0.85	0.418975	0.884870
10	1	50	50	0.9	0.431496	0.793640
10	1	50	50	0.95	0.442035	0.659470
10	1	50	50	1	0.446189	0.549699
10	1	50	100	0	0.393275	1.000000
10	1	50	100	0.5	0.393275	1.000000
10	1	50	100	0.55	0.393284	0.999989
10	1	50	100	0.6	0.393303	0.999962
10	1	50	100	0.65	0.393716	0.999265
10	1	50	100	0.7	0.396377	0.993596
10	1	50	100	0.75	0.402283	0.977821
10	1	50	100	0.8	0.411470	0.945201
10	1	50	100	0.85	0.421230	0.897748
10	1	50	100	0.9	0.432730	0.813671
10	1	50	100	0.95	0.442351	0.690785
10	1	50	100	1	0.446189	0.588397
10	1	50	200	0	0.403272	1.000000
10	1	50	200	0.5	0.403272	1.000000
10	1	50	200	0.55	0.403280	0.999992
10	1	50	200	0.6	0.403288	0.999980
10	1	50	200	0.65	0.403500	0.999620
10	1	50	200	0.7	0.405194	0.996005
10	1	50	200	0.75	0.409135	0.985464
10	1	50	200	0.8	0.415928	0.961331
10	1	50	200	0.85	0.423850	0.922681
10	1	50	200	0.9	0.433688	0.850310
10	1	50	200	0.95	0.442615	0.735373
10	1	50	200	1	0.446189	0.638831
10	1	50	300	0	0.389349	1.000000
10	1	50	300	0.5	0.389349	1.000000
10	1	50	300	0.55	0.389359	0.999989
10	1	50	300	0.6	0.389363	0.999983
10	1	50	300	0.65	0.389636	0.999522
10	1	50	300	0.7	0.391365	0.995818
10	1	50	300	0.75	0.395754	0.984073
10	1	50	300	0.8	0.404160	0.954127
10	1	50	300	0.85	0.413674	0.907826
10	1	50	300	0.9	0.427072	0.809817
10	1	50	300	0.95	0.440501	0.639990
10	1	50	300	1	0.446189	0.494129
100	0	10	50	0	0.281941	1.000000
100	0	10	50	0.5	0.281941	1.000000
100	0	10	50	0.55	0.281941	1.000000
100	0	10	50	0.6	0.281941	1.000000
100	0	10	50	0.65	0.281941	1.000000
100	0	10	50	0.7	0.281944	0.999994
100	0	10	50	0.75	0.282068	0.999643
100	0	10	50	0.8	0.282948	0.996548
100	0	10	50	0.85	0.287040	0.976778
100	0	10	50	0.9	0.296715	0.909163
100	0	10	50	0.95	0.302578	0.834636
100	0	10	50	1	0.308204	0.619209
100	0	10	100	0	0.212147	1.000000
100	0	10	100	0.5	0.212147	1.000000

100	0	10	100	0.55	0.212447	0.999668
100	0	10	100	0.6	0.215474	0.995449
100	0	10	100	0.65	0.222441	0.983695
100	0	10	100	0.7	0.235223	0.956751
100	0	10	100	0.75	0.254306	0.905973
100	0	10	100	0.8	0.273850	0.839286
100	0	10	100	0.85	0.285619	0.783696
100	0	10	100	0.9	0.295544	0.712626
100	0	10	100	0.95	0.304316	0.600746
100	0	10	100	1	0.308204	0.488648
100	0	10	200	0	0.243530	1.000000
100	0	10	200	0.5	0.243530	1.000000
100	0	10	200	0.55	0.243589	0.999939
100	0	10	200	0.6	0.244546	0.998583
100	0	10	200	0.65	0.247376	0.993787
100	0	10	200	0.7	0.254668	0.978326
100	0	10	200	0.75	0.264269	0.952815
100	0	10	200	0.8	0.276121	0.912003
100	0	10	200	0.85	0.285872	0.865412
100	0	10	200	0.9	0.294678	0.801949
100	0	10	200	0.95	0.303739	0.684533
100	0	10	200	1	0.308204	0.555209
100	0	10	300	0	0.220990	0.747475
100	0	10	300	0.5	0.220990	0.747475
100	0	10	300	0.55	0.221099	0.747349
100	0	10	300	0.6	0.222519	0.745360
100	0	10	300	0.65	0.226636	0.738422
100	0	10	300	0.7	0.236331	0.717926
100	0	10	300	0.75	0.256533	0.663618
100	0	10	300	0.8	0.279104	0.586903
100	0	10	300	0.85	0.289505	0.538252
100	0	10	300	0.9	0.298203	0.476328
100	0	10	300	0.95	0.305451	0.385300
100	0	10	300	1	0.308204	0.306653
100	0	30	50	0	0.282217	1.000000
100	0	30	50	0.5	0.282217	1.000000
100	0	30	50	0.55	0.282217	1.000000
100	0	30	50	0.6	0.282217	1.000000
100	0	30	50	0.65	0.282217	1.000000
100	0	30	50	0.7	0.282219	0.999995
100	0	30	50	0.75	0.282355	0.999609
100	0	30	50	0.8	0.283216	0.996590
100	0	30	50	0.85	0.287243	0.977147
100	0	30	50	0.9	0.296813	0.910079
100	0	30	50	0.95	0.302736	0.834857
100	0	30	50	1	0.308204	0.625029
100	0	30	100	0	0.211958	1.000000
100	0	30	100	0.5	0.211958	1.000000
100	0	30	100	0.55	0.212246	0.999681
100	0	30	100	0.6	0.215279	0.995448
100	0	30	100	0.65	0.222211	0.983751
100	0	30	100	0.7	0.235089	0.956597
100	0	30	100	0.75	0.254156	0.905908
100	0	30	100	0.8	0.273772	0.838959
100	0	30	100	0.85	0.285640	0.782929
100	0	30	100	0.9	0.295429	0.712932
100	0	30	100	0.95	0.304348	0.599493
100	0	30	100	1	0.308204	0.488356
100	0	30	200	0	0.242816	1.000000
100	0	30	200	0.5	0.242816	1.000000
100	0	30	200	0.55	0.242873	0.999941
100	0	30	200	0.6	0.243839	0.998575
100	0	30	200	0.65	0.246693	0.993741

100	0	30	200	0.7	0.254200	0.977819
100	0	30	200	0.75	0.263913	0.951992
100	0	30	200	0.8	0.275886	0.910760
100	0	30	200	0.85	0.285639	0.864184
100	0	30	200	0.9	0.294561	0.799843
100	0	30	200	0.95	0.303739	0.680746
100	0	30	200	1	0.308204	0.551326
100	0	30	300	0	0.195125	0.898990
100	0	30	300	0.5	0.195125	0.898990
100	0	30	300	0.55	0.195357	0.898724
100	0	30	300	0.6	0.198153	0.894820
100	0	30	300	0.65	0.205386	0.882627
100	0	30	300	0.7	0.220365	0.851093
100	0	30	300	0.75	0.247797	0.777752
100	0	30	300	0.8	0.273374	0.690933
100	0	30	300	0.85	0.285819	0.632586
100	0	30	300	0.9	0.296297	0.557959
100	0	30	300	0.95	0.304923	0.449441
100	0	30	300	1	0.308204	0.355723
100	0	50	50	0	0.280291	1.000000
100	0	50	50	0.5	0.280291	1.000000
100	0	50	50	0.55	0.280291	1.000000
100	0	50	50	0.6	0.280291	1.000000
100	0	50	50	0.65	0.280291	1.000000
100	0	50	50	0.7	0.280294	0.999994
100	0	50	50	0.75	0.280445	0.999564
100	0	50	50	0.8	0.281435	0.996090
100	0	50	50	0.85	0.285840	0.974880
100	0	50	50	0.9	0.296277	0.902145
100	0	50	50	0.95	0.302437	0.823923
100	0	50	50	1	0.308204	0.604591
100	0	50	100	0	0.207525	1.000000
100	0	50	100	0.5	0.207525	1.000000
100	0	50	100	0.55	0.207861	0.999628
100	0	50	100	0.6	0.211136	0.995062
100	0	50	100	0.65	0.218551	0.982552
100	0	50	100	0.7	0.232054	0.954102
100	0	50	100	0.75	0.252591	0.899394
100	0	50	100	0.8	0.273227	0.829027
100	0	50	100	0.85	0.285417	0.771496
100	0	50	100	0.9	0.295373	0.700221
100	0	50	100	0.95	0.304276	0.586873
100	0	50	100	1	0.308204	0.474323
100	0	50	200	0	0.241403	1.000000
100	0	50	200	0.5	0.241403	1.000000
100	0	50	200	0.55	0.241462	0.999938
100	0	50	200	0.6	0.242457	0.998530
100	0	50	200	0.65	0.245560	0.993269
100	0	50	200	0.7	0.253189	0.977083
100	0	50	200	0.75	0.263246	0.950351
100	0	50	200	0.8	0.275484	0.908215
100	0	50	200	0.85	0.285423	0.860762
100	0	50	200	0.9	0.294451	0.795723
100	0	50	200	0.95	0.303713	0.676003
100	0	50	200	1	0.308204	0.545665
100	0	50	300	0	0.176418	1.000000
100	0	50	300	0.5	0.176418	1.000000
100	0	50	300	0.55	0.176853	0.999514
100	0	50	300	0.6	0.181430	0.993144
100	0	50	300	0.65	0.191605	0.975987
100	0	50	300	0.7	0.209955	0.937468
100	0	50	300	0.75	0.239940	0.857827
100	0	50	300	0.8	0.266520	0.767340

100	0	50	300	0.85	0.281556	0.696474
100	0	50	300	0.9	0.294628	0.603525
100	0	50	300	0.95	0.304613	0.478567
100	0	50	300	1	0.308204	0.376334
100	1	10	50	0	0.306915	1.000000
100	1	10	50	0.5	0.306915	1.000000
100	1	10	50	0.55	0.306915	1.000000
100	1	10	50	0.6	0.306915	1.000000
100	1	10	50	0.65	0.306915	1.000000
100	1	10	50	0.7	0.306917	0.999996
100	1	10	50	0.75	0.307026	0.999688
100	1	10	50	0.8	0.307888	0.996663
100	1	10	50	0.85	0.312413	0.974872
100	1	10	50	0.9	0.321241	0.913813
100	1	10	50	0.95	0.327557	0.831933
100	1	10	50	1	0.333865	0.601131
100	1	10	100	0	0.238047	1.000000
100	1	10	100	0.5	0.238047	1.000000
100	1	10	100	0.55	0.238292	0.999733
100	1	10	100	0.6	0.241184	0.995684
100	1	10	100	0.65	0.247752	0.984561
100	1	10	100	0.7	0.261245	0.956075
100	1	10	100	0.75	0.279194	0.908391
100	1	10	100	0.8	0.296308	0.849945
100	1	10	100	0.85	0.308492	0.791833
100	1	10	100	0.9	0.319788	0.710768
100	1	10	100	0.95	0.330076	0.579494
100	1	10	100	1	0.333865	0.472296
100	1	10	200	0	0.269697	1.000000
100	1	10	200	0.5	0.269697	1.000000
100	1	10	200	0.55	0.269739	0.999957
100	1	10	200	0.6	0.270547	0.998810
100	1	10	200	0.65	0.272945	0.994741
100	1	10	200	0.7	0.279803	0.980174
100	1	10	200	0.75	0.288808	0.956246
100	1	10	200	0.8	0.299394	0.919693
100	1	10	200	0.85	0.309269	0.872146
100	1	10	200	0.9	0.319052	0.801527
100	1	10	200	0.95	0.329582	0.664437
100	1	10	200	1	0.333865	0.542452
100	1	10	300	0	0.285144	0.505051
100	1	10	300	0.5	0.285144	0.505051
100	1	10	300	0.55	0.285171	0.505019
100	1	10	300	0.6	0.285607	0.504403
100	1	10	300	0.65	0.287204	0.501699
100	1	10	300	0.7	0.291891	0.491695
100	1	10	300	0.75	0.301385	0.466252
100	1	10	300	0.8	0.312328	0.429156
100	1	10	300	0.85	0.318712	0.398885
100	1	10	300	0.9	0.325764	0.348250
100	1	10	300	0.95	0.331758	0.272873
100	1	10	300	1	0.333865	0.213621
100	1	30	50	0	0.304407	1.000000
100	1	30	50	0.5	0.304407	1.000000
100	1	30	50	0.55	0.304407	1.000000
100	1	30	50	0.6	0.304407	1.000000
100	1	30	50	0.65	0.304407	1.000000
100	1	30	50	0.7	0.304410	0.999994
100	1	30	50	0.75	0.304548	0.999603
100	1	30	50	0.8	0.305514	0.996203
100	1	30	50	0.85	0.310588	0.971788
100	1	30	50	0.9	0.320389	0.904152
100	1	30	50	0.95	0.327328	0.814129

100	1	30	50	1	0.333865	0.576664
100	1	30	100	0	0.235253	1.000000
100	1	30	100	0.5	0.235253	1.000000
100	1	30	100	0.55	0.235495	0.999734
100	1	30	100	0.6	0.238593	0.995403
100	1	30	100	0.65	0.245514	0.983690
100	1	30	100	0.7	0.259479	0.954256
100	1	30	100	0.75	0.278689	0.903096
100	1	30	100	0.8	0.296078	0.843669
100	1	30	100	0.85	0.308483	0.784499
100	1	30	100	0.9	0.319828	0.703087
100	1	30	100	0.95	0.330056	0.572624
100	1	30	100	1	0.333865	0.465483
100	1	30	200	0	0.269403	1.000000
100	1	30	200	0.5	0.269403	1.000000
100	1	30	200	0.55	0.269442	0.999959
100	1	30	200	0.6	0.270223	0.998854
100	1	30	200	0.65	0.272659	0.994716
100	1	30	200	0.7	0.279706	0.979745
100	1	30	200	0.75	0.288718	0.955788
100	1	30	200	0.8	0.299264	0.919388
100	1	30	200	0.85	0.309228	0.871441
100	1	30	200	0.9	0.319007	0.800823
100	1	30	200	0.95	0.329602	0.663133
100	1	30	200	1	0.333865	0.541647
100	1	30	300	0	0.209223	0.898990
100	1	30	300	0.5	0.209223	0.898990
100	1	30	300	0.55	0.209459	0.898721
100	1	30	300	0.6	0.212566	0.894374
100	1	30	300	0.65	0.221491	0.879264
100	1	30	300	0.7	0.240732	0.838780
100	1	30	300	0.75	0.271634	0.756567
100	1	30	300	0.8	0.294986	0.677472
100	1	30	300	0.85	0.308586	0.613031
100	1	30	300	0.9	0.321683	0.519787
100	1	30	300	0.95	0.330897	0.404695
100	1	30	300	1	0.333865	0.321514
100	1	50	50	0	0.304758	1.000000
100	1	50	50	0.5	0.304758	1.000000
100	1	50	50	0.55	0.304758	1.000000
100	1	50	50	0.6	0.304758	1.000000
100	1	50	50	0.65	0.304758	1.000000
100	1	50	50	0.7	0.304762	0.999992
100	1	50	50	0.75	0.304909	0.999576
100	1	50	50	0.8	0.305901	0.996097
100	1	50	50	0.85	0.310885	0.972121
100	1	50	50	0.9	0.320599	0.905027
100	1	50	50	0.95	0.327305	0.818101
100	1	50	50	1	0.333865	0.578585
100	1	50	100	0	0.233534	1.000000
100	1	50	100	0.5	0.233534	1.000000
100	1	50	100	0.55	0.233794	0.999715
100	1	50	100	0.6	0.237033	0.995186
100	1	50	100	0.65	0.244081	0.983248
100	1	50	100	0.7	0.258284	0.953319
100	1	50	100	0.75	0.278077	0.900600
100	1	50	100	0.8	0.295520	0.840982
100	1	50	100	0.85	0.308077	0.781156
100	1	50	100	0.9	0.319701	0.697718
100	1	50	100	0.95	0.329995	0.566573
100	1	50	100	1	0.333865	0.458025
100	1	50	200	0	0.268280	1.000000
100	1	50	200	0.5	0.268280	1.000000

100	1	50	200	0.55	0.268319	0.999959
100	1	50	200	0.6	0.269213	0.998693
100	1	50	200	0.65	0.271740	0.994399
100	1	50	200	0.7	0.278909	0.979174
100	1	50	200	0.75	0.288170	0.954554
100	1	50	200	0.8	0.298971	0.917264
100	1	50	200	0.85	0.308939	0.869280
100	1	50	200	0.9	0.318811	0.798010
100	1	50	200	0.95	0.329568	0.658380
100	1	50	200	1	0.333865	0.535500
100	1	50	300	0	0.201423	1.000000
100	1	50	300	0.5	0.201423	1.000000
100	1	50	300	0.55	0.201767	0.999621
100	1	50	300	0.6	0.206143	0.993504
100	1	50	300	0.65	0.216053	0.976717
100	1	50	300	0.7	0.235860	0.935098
100	1	50	300	0.75	0.264379	0.859462
100	1	50	300	0.8	0.287457	0.780658
100	1	50	300	0.85	0.304102	0.701796
100	1	50	300	0.9	0.319952	0.589237
100	1	50	300	0.95	0.330480	0.457646
100	1	50	300	1	0.333865	0.362631