Quran Analytics with R

A Practical Introduction

Wan M Hasni and Azman Hussin

2024-03-22

Contents

Acknowledgements

We wish to acknowledge the **R** programming community for their contributions to many wonderful packages which are well documented and supported by the developers. The packages are a significant contribution to the work of **data science**.

We also acknowledge *RPubs* which is an open publishing platform for HTML documents produced using RMarkdown from within RStudio, which we use to publish our blog articles.

We also acknowledge **R** package *bookdown* through which the formatting and integrating with *Latex* and preparations for this book is possible. Of course, a special thanks to the author of the **quRan** package, without which we would not have started this project.

We thank our families for their enduring patience and support.

As always, the truth is only from Allah, and all errors are solely ours. Lastly, we submit our gratitude to Allah (SWT) knowing that we can never thank Him (SWT) enough for having helped us in this project.

Preface From the First Author

> "Nun. By the Pen (Al-Qalam) and what they inscribe" [Al-Qalam:1]

My journey into the Quran and Islamic knowledge started way back during the university days through the self-learning of Arabic while I was doing my first degree in Statistics and Actuarial Science. I was engaged with data and numbers, statistics and modeling, and applications in risk studies, economic, and finance during my student years until the Ph.D. Today, the field that I was engaged in is called "Applied Data Science". The term data science was nonexistent then. During those days, I took the study of Al-Quran and Islamic knowledge as a private passion. Applying knowledge to Islamic finance was my main preoccupation.

What struck me then was how easy it was to learn Arabic, particularly if the interest is Classical Arabic or Quranic Arabic. Arabic is an extremely structured language; once you grasp the structure, the rest is just about acquiring more vocabularies and meaning or contexts. I relied heavily on the texts available then, namely the abridged version of Tafseer Ibnu Kathir, which came in three volumes, and Hans-Wehr Arabic-English dictionary. Within a short time, I can read most of the Classical Arabic texts with little problem. However, despite all of the effort, whatever I do is only to benefit my knowledge and enrichment.

I know that to be well versed as a scholar in the field is way beyond my reach, and I was deeply engaged in my professional works at all times.

My data science journey continued as I discovered the **R** programming language in its early days in 2001 when I worked on economic data. What strikes me was many functionalities of R are similar to what I was working with at the University of Iowa, namely the Minitab statistical program and the Wolfram Mathematica. Moreover, to a large degree, R has similarities to the C++ program, the primary programming language I was using for my doctorate research work.

From 2010 onwards, I saw the rise of Machine Learning and Artificial Intelligence as a new field. I then took a deep dive into ML and AI algorithms, which took me to various fields of Statistical Learning, Machine Learning, Neural Networks, Deep Learning, and finally into Graph Theory and Network Science. In this journey, I refresh myself back into several prime areas in my doctorate days, namely in Game Theory, Behavioral Economics, and Systems Theory. Over the last decade, my occupation is in Data Science, and apply them within market applications under Data Analytics and Artificial Intelligence. While doing all of these, Data Science and Artificial Intelligence applications on Islamic knowledge have always been my penultimate target.

Usage of modern tools to uncover or rediscover the Truth is an apparent necessity. We may not be able to produce a new Imam Al-Shafi'e or Imam Al-Bukhari, but we may be able to replicate some of what they had accomplished. However, this is possible (at least for now) by relooking the Islamic sciences using new lenses. These new lenses are what we are starting to do here. Namely, to start a new work on Islamic Knowledge Analytics, beginning with Quran Analytics. I start this with the verse "Nun. By the Pen (Al-Qalam) what they inscribe".

As in anything else, the starting point is the hardest; and I know that I cannot do it alone. It must be based on many people's collaborative network, from many domains of expertise, regardless of their creed or belief, as long as the Truth is what he/she is after. I thank Dr. Azman Hussin, my co-author for joining me in this work. His encouragement is extremely appreciated.

I welcome you on board this journey. The reward is from the owner of the Truth when you meet Him.

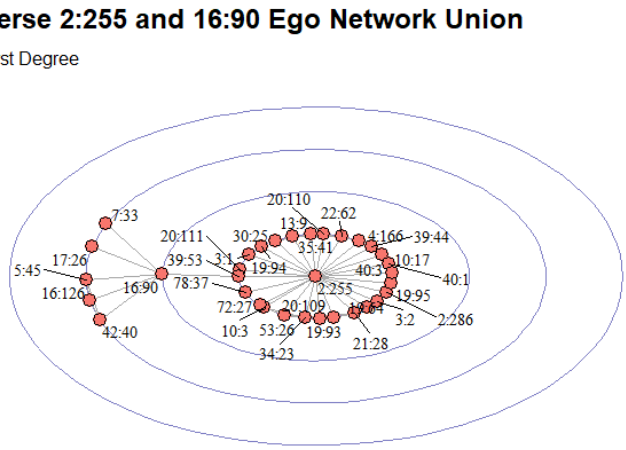
Preface From the Second Author

"The best of you are those who learn the Quran and teach it."

There is no other motivation that drives this humble work except the above saying of Prophet Muhammad (SAW). It is an honor to give an added dimension to "learning" the Quran through data analytics. It is also proof of the eternal relevance of the Quran as a book of knowledge in that using new tools like text analytics reveals new knowledge and understanding about the Quran. The second part of the above saying motivated us to document our work and make it freely available in digital form. In keeping with the tradition of the prestigious scholars of the Quran, we humbly present our work as a gift to the world, particularly the "data generation". We sincerely hope others will improve upon this work.

We formally started our #qurananalytics project to accompany our #learnbydoing approach to #networkscience by defining some objectives and methods.[1] The main trigger was when we came across a well prepared dataset of the Holy Quran which was quite recently published. The obvious idea came to apply the current tools and methods in text analysis with the Quran datasets. In research, different methods used on the same data often yield different results. So, we thought that using the new methods and tools of #datascience, #textanalytics, and #networkscience to "learn the Quran" should be interesting. The Quran is also a subject matter that we have some knowledge of since both of us have a working knowledge of written Arabic and continue to deepen and broaden our own understanding of the Quran. It seems an ideal combination to build knowledge of the tools while applying them to a familiar subject matter.

Later on, the learning and encouraging feedback we got from the blog postings led us to further explore the application of these tools on the classical works of Tafseer, the interpretation of the Quran. We had some ideas how #networkscience tools can help us visualize the classical method of Tafseer al-Quran bi al-Quran (Interpretation of the Quran with the Quran). It is considered the best method to interpret the verses of the Quran. We showed some features of network graphs that enable us to visualize aspects of the Tafseer that is almost impossible to do manually and difficult to do without a network graph model. We discuss this in Chapter 9 of this book where we present our preliminary findings using network analysis applied to the best of the classical works of Tafseer, Tafseer Ibnu Katheer. Simple network concepts like "in-degrees", "out-degrees" and "ego networks" proved valuable in analyzing Tafseer Ibnu Katheer.



Along the way, we also developed some customized tutorials on the tools that we used. These tutorials cover selected individual Surahs (Chapters) of the Quran like Surah Yusuf, Surah Taa Haa, and Surah Al-Kahfi - all of which are covered in different Chapters of the book. We hope our readers can learn the tools within the subject of "learning the Quran".

In the initial project brief[2], we wrote under the "Objectives" section,

[1] Introduction to Quran Analytics, https://www.linkedin.com/pulse/introduction-quran-analytics-azman-hussin
[2] Introduction to Quran Analytics, https://www.linkedin.com/pulse/introduction-quran-analytics-azman-hussin

We hope that this work will result in a book or manual on how to do Quran Analytics.

Alhamdulillah, the hope became a reality. I sincerely thank my co-author, Dr. Wan Hasni, for introducing me to #networkscience and lending me the many books he had on the subject. He painstakingly corrected and converted the R Markdown files I passed to him to be part of the contents of the book. I also would like to thank the reviewers for their valuable comments.

This book is designed as an introductory manual on #qurananalytics using R. It includes customized tutorials on the tools we used, sample codes, data wrangling techniques, and analysis of the results. This will help others interested in the subject to pick up the related skills and knowledge, comment on the findings, and to develop it further. In the final chapter, we included a "to-do" list that interested readers can further explore.

We can only express our sincere gratitude to our Lord and Creator for giving us this honor to add a drop in the ocean of knowledge related to His Book. Our salutations to the beloved Prophet Muhammad (SAW) whose saying motivated us in this work. We pray that our readers benefit from this humble gift and continue adding value to the chain of the Quranic knowledge.

"And our Lord is the Beneficent Allah, Whose help is sought" [Quran, 21:112]

Preamble

This book originates from a compilation of joint works by both authors published in RPubs under (https://rpubs.com/azmanH/). The works began as an exploratory exercise, which emanates from our interest in #networkscience which lead us to #qurananalytics. We admit that this is a new field that is relatively less explored. It requires domain expertise in linguistics, particularly language in Islamic religious texts, data programming language, statistics and machine learning, and Natural Language Processing (NLP). Furthermore, the Quran is the sacred religious text for Muslims, which makes the task requires careful considerations and treatment.

We decided to compile them into a book while knowingly that the work is far from complete and at best still at exploratory stages. It dawns on us that even in exploratory stages, the amount of works required to complete the project is vast and requires massive collaboration efforts by people from various domains of expertise. The fastest way to get the interest and others to join us in this journey is to publish our works thus far and present them to as wide an audience as possible, through a book.

In this book, which we plan to be a series of books in the future, we will present an introduction to the subject of Quran Analytics, with focuses on using tools of NLP, deploying **R** programming language. The purpose is threefold, introducing the subject of Quran Analytics, exploring NLP tools, and use the R programming environment as the tool of choice.

Our wish is for many other researchers to follow our track, collaborate with us in our efforts - especially for people with expertise or knowledge in one of the domains, but lacking expertise in the other domain. The journey is not that hard if a person already knows one of the domains. For example, many experts in the sciences of Al Quran do not have exposure to the computing environment, and hence do not understand NLP deeply; our argument is NLP is first and foremost is about linguistics, cast within computational linguistics. As far as **R** programming language, despite being an uphill task for beginners to be familiar with its working environment, over the years of development, it had been made easy with many recent works on it, such as the **R Studio** environment as its IDE. It is just like riding a bicycle, if you know how to ride one, then almost any bicycle is the same. R is among the simplest bicycle around.

We plan this book to be an open-source concept. It is prepared using the bookdown package in R (https://bookdown.org/yihui/bookdown/). All the data and codes will available at in a **github** site to be added in the future. Some of our future works will be posted on the RPubs site (https://rpubs.com/azmanH/). Eventually, we will also produce some packages in R, which will be posted on CRAN.

1 Introducing Quran Analytics

> Prophet Muhammad (SAW) said, "The best of you are those who learn the Quran and teach it." [Bukhari]

Learning and teaching Al-Quran is one of the greatest virtues in Islam. It is an obligatory duty of a Muslim. The methods of learning Al-Quran have been passed down from generation to generation and continue to flourish even to this day. The methods of learning Al-Quran had always been in the original text of Al-Quran, which is the Quranic Arabic; whether it is about learning to read or learning to understand.

With advancements in technology, is it possible for the new generation to learn Al-Quran beyond the traditional methods and tools? Does the digital revolution allow us to use technologies for learning and teaching Al-Quran? The answer is most clearly a yes. However, people with knowledge of technology understand that the use of technology is beyond just browsing webpages and manipulating mobile applications within our devices. It is also beyond simple webpage creation and publication of YouTube videos. All these means of teaching and learning, while laudable, bring along many unintended consequences.

The question we want to pose is, is it possible to put everything about teaching and learning Al-Quran as a comprehensive and all-encompassing system using digital technology of the day? If this quest is possible, then we have to start the subject right from the very beginning - that is to cast the objective of the envisaged system within the domain of scientific discovery.

The logic of knowledge (or scientific discovery) provides an analysis of testing systems of theories against experience by observation and experiment (Popper, 1992). Analyzing is one mechanism to produce reliability based on evidence. Evidence comes in the form of data, and testing is through inference. Inference, on the other hand, is understood through the theory of probability.

The question is, can we put the same rigor of testing to Al-Quran as a way for us to get to the truth or true belief? Al-Quran to Muslims is a sacred religious text, the "word of Allah," a divine revelation, and is the final truth. For many Muslims, Al-Quran studies must be through traditional learning and knowledge discovery methods. First and foremost, we hold to the view that as Muslims, believing in the finality of the truth of Al-Quran is one of the pillars of our faith; and there is no question about it. Therefore, we believe that the question of "testing" the truth of Al-Quran does not arise; in fact, it should reaffirm any believer in the veracity of its truth.

The basic principle must be that what we seek is knowledge itself. Knowledge and discovery of knowledge do not have any limit. As a believer, we must have a firm believe in the miraculousness of the language of Al-Quran; but to be able to appreciate this miraculousness with reason is what we lack. Without formal training in the Classical Arabic language and Islamic studies, many of us are at the mercy of those who claimed authority over the subject. Is there a way for those in this category to acquire this knowledge using modern tools, directly from the original and classical sources of the Quranic knowledge? Answering this is what Quran Analytics is all about.

Modern tools in knowledge discovery and acquisition are computational methods, data science, artificial intelligence, and algorithms - combined with scientific methods of analysis and domain expertise - applied in computing environments. Examples of this are information retrieval, knowledge extractions, knowledge graphs, and many other applications in various fields of studies in natural and social sciences. The web, as we know it today, is a massive body of information and knowledge. Extracting knowledge from the web is impossible without relying on these modern tools.

Quranic knowledge is vast from many aspects; "the language is a captivating beauty, boundless expressiveness, eloquence, rhythm, magic, and distinction."(Saeh, 2015) It benefited from the "turath" of many scholars of the past with voluminous texts. These scholars came across many geographic regions of the world over more than one thousand years. The question is, how do we gain knowledge from this vast sea of knowledge? The answer is, we need modern tools for knowledge acquisition in the form of Quran Analytics.

The earlier scholars of Islam's approach to Quranic knowledge were eminently clear. They start with a thorough mastery of the language, and secondly through the acquisition of knowledge via memorization of everything in its original form (i.e., language in words and speech, as well as in context) and from total comprehension of knowledge through understanding the various teachings. However, as time progresses, the subject of language is becoming more distant, especially for many of the non-Arabic speaking people (due to the language barrier) and Arabic speaking people (due to diglossic behavior). The distance of Quranic knowledge from the people is growing.

Furthermore, the path of knowledge development among the early Islamic scholars took different trajectories; for example, there are distinct paths taken among scholars from the Arabic peninsula, the Levant, Central Asia, North Africa, the Turk regions, the sub-continent, and Southeast Asia. As the knowledge is being transmitted and spread, many changes happened, and the eventual form of understanding may have diverged significantly. An example of this is the Mazhabs (sects), wherein Southeast Asia ascribes to Shafie, the Persians to Shi'a, the Turks to Hanafi. Tracing these various paths of knowledge development by itself is a significant task, not counting the content and context of the knowledge itself.

Written texts possess something that others do not, namely, it is a form of data and carries the language with the contexts and meaning and knowledge dissemination. With advancements in computing, these texts are available in digitized form and further digitalized in a machine-readable format. With the advancements in the tools of annotation and textual analysis together with Big Data availability, the process of analyzing becomes less daunting.

Quran Analytics is a new term without a clear scholarly definition. We know that it is a combination of various disciplines - including Quranic sciences (Ulum Al-Quran), language and linguistics, and broader Islamic sciences. The relationship and interdependencies between these fields are both *complicated* and *complex*. Our exploratory attempt to provide a diagrammatic relation between the various fields of knowledge is shown in Figure 1.

The "relationships" shown are examples of the relations and intersections of multi-disciplinary requirements for Quran Analytics.

The vast subject of Quran Analytics is our broad domain. We cannot cover all in the current work. Instead, in this book, we would like to start introducing Quran Analytics, focusing on selected areas. We intend to start and spur the development of Quran Analytics as a new field of study, and this book is our introduction to the subject.

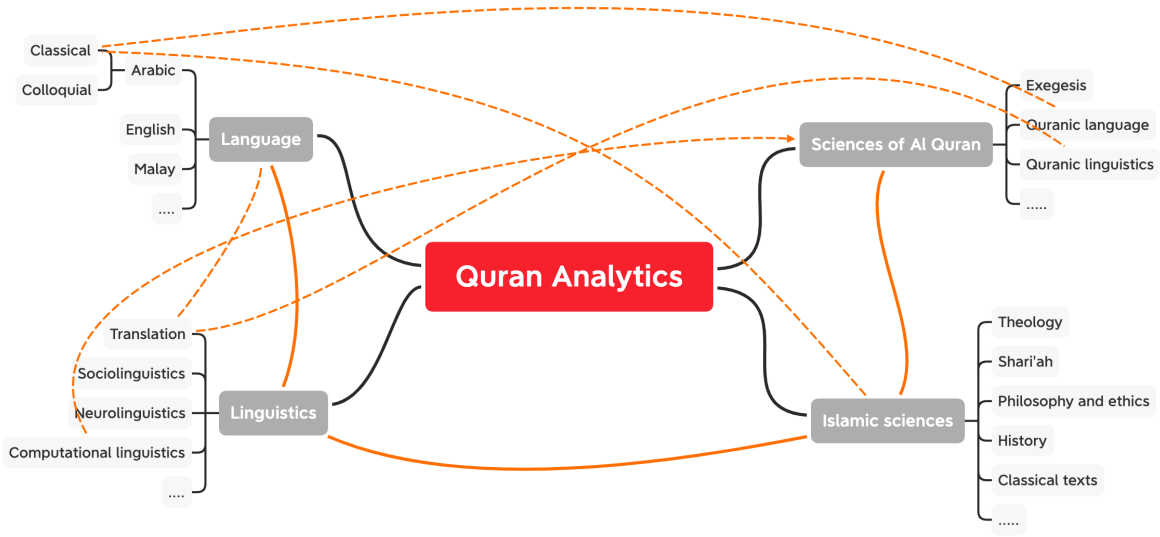


Figure 1: Complexity of Quran Analytics

1.1 Quran Analytics

Our search for the term "quran analytics" in Google Scholar, yields 2,480 results; and a similar search for "bible analytics" yields 19,500 results.[3] In both cases, we could not find a clear scholarly definition of the terms used. In the case of the Bible, for example, one interesting site we found is Aver Analytics,[4] where they perform annotations of the King James (English version) Bible and convert the text data into visuals, for the purpose of learning. For the case of Quran Analytics, the main site is from qurananalysis,[5] which is the product of research works by the School of Computing team at the University of Leeds, United Kingdom. The site claims to be "the swiss-knife of Quran Research" for both Arabic and English languages.

The most comprehensive reference that we found on the subject is based on the works of Prof. Eric Atwell and his students at Leeds University. Over a decade, they have been working on annotating the Quranic texts in Arabic, using the Buckwalter transliteration engines for Arabic textual language, generating various statistical measures for the Quranic words, and creating meta-data for words and verses of the Quran. While there are many recent scholarly articles such as publications of the peer-reviewed Journal of Qur'anic Studies by the University of Edinburgh, Quranic Analytics is a subject not extensively covered; whether it is labeled as "Quran NLP", "Quran Analytics", or anything similar. After reading and scanning the works, we still could not find definitive definitions of Quran Analytics.

In the absence of a clearly defined terminology, we propose that the term, Quran Analytics, means the works of combining the science of Quranic studies, language and linguistic studies, and computational linguistics, for the knowledge acquisition and extraction from the Quran.

Defining exactly what is meant by Quran Analytics will be a major challenge; a subject which we intend to develop and expand extensively in our research. For now, we would say that our approach to Quran Analytics is about utilizing the tools of computational linguistics applied non-parametrically to the studies of the sciences of Al-Quran. What we mean by non-parametric is that, by design, the analysis does not assume any pre-existing models of the sciences of Al-Quran. Another way to express this is akin to the terminology

[3]Search performed on 27 January 2021, using https://scholar.google.com/
[4]https://www.averanalytics.com/bible-dashboard
[5]https://www.qurananalysis.com/analysis/

of "unsupervised learning" in machine learning algorithms. And borrowing the term from network or graph theory, we perform analysis to allow self-emergent structures to emerge and be detected in the process.

Where this approach will take us to is still uncertain. However, the encouraging development is, for many complex tasks in computational linguistics such as information retrieval and knowledge extractions, using unsupervised learning, knowledge graphs (both are non-parametric methods), are among the major discoveries of recent time. This is largely aided by the availability of Big Data, faster computing capabilities, accuracies of newer algorithms, and advancements in network science.

1.2 Quranic Studies

Traditional studies of Al-Quran are through the Arabic language, the science of Quranic studies (Ulum al-Quran), and the exegesis (Tafseer) of Al-Quran. The methods go back to the early period of Islam and are passed down to modern times. It is an extensively developed field which in early days were dominated only in the Arabic language, until recent times when translations to other languages such as English were made. There exist large corpora of text, in Arabic, for various exegesis, commentaries, etc., aggregated from more than one thousand years of development. All of the writings have been preserved and nowadays available in digital forms.

Anyone studying Al-Quran knows that to be able to read and comprehend all possible exegesis and commentaries for the Quran is a major task even for a scholar in the field. For people without proficiency in the Arabic language, the task is even harder. Only recently, for example, Tafseer Ibn Kathir is translated to English and other languages from its Arabic origin. Translating is not an easy effort, since it is often subjected to the translators' own views and biases.

One of the approaches to Quranic studies is to express it as literary texts within linguistic contexts. While the approach is novel, it is highly criticized by some traditional Islamic scholars as violations of Islamic principles, since the Quran is the "word of God" and not some literary piece. There is a lot of truth in this criticism, especially when it is studied in non-Quranic Arabic forms of language, which includes colloquial Arabic and non-Arabic languages. Furthermore, as emphasized by early scholars of the sciences of Al-Quran, the knowledge from the Quran must be viewed as a holistic approach, rather than analyzing their grammar, lexicography, content, and implications within sentences (verses) and a group of sentences (verses).

In this regard, the arcane Computational Linguistic (CL) approach to Quranic studies, will be highly criticized since it might not adhere to the established methods and rules. Therefore there is a serious need for **combining the sciences of Quranic studies with CL**, whereby CL is only used as a tool for performing the studies, and not the primary method. In other words, CL on Al-Quran requires first and foremost, the sciences of Quranic studies as a field of knowledge and then CL as a tool of computing and processing the texts.

Our view is that, to date, most of the incomprehension of CL applications to the Quranic studies comes from the lack of understanding of scholars of the CL subject itself. CL is only good as a tool, which follows the rules imposed by the designer or developer of the tools. Properly designed tools will serve their purpose. An example of this is the application of CL tools for the Chinese language and literary texts. Developers of Chinese CL tools have to reconstruct the engines of their CL application to perform with high accuracy. In other words, any computing solution is not "model-free" and therefore requires an intense modeling process. The sciences of Quranic studies and the science of Quranic Arabic from classical sources are a good start to derive these CL rules. This field has not been extensively developed, and hence we intend to expand the research and development of the applications of CL to the science of Quranic studies.

1.3 Quranic language and linguistic studies

Language studies is a very old field, as old as humanity itself. Language as a subject is difficult to be described in simple terms, as it is inter-related with many fields of literature, philology, anthropology, sociology, as well as biology. The Quranic language furthermore is unique in the sense that, we believe that it is the word of

Allah, revealed through Prophet Muhammad (SAW), in Arabic language, one thousand two hundred years ago. As a language, traditional and contemporary scholars have argued that the Quranic language stands alone, apart even from the Classical Arabic of the time, and distinctly clear from the Colloquial Arabic.

Our position is that the Quranic language has to be the first priority in the pecking order. It should be the rule which governs the meaning of the language. As an example, the Quranic language uses the "past tense" in English meaning something that is perpetual (e.g. the word "kaana"), in which case, its common usage of Arabic grammar (Classical or Colloquial) will never be the same as Quranic Arabic. Quran also introduces new terms and names or gives meanings to these terms and names. The same term and meanings do not change in Quranic Arabic but may change in the usage in the society (i.e. natural language). An example of this is "Riba". A recent reference on the subject is found in (Saeh, 2015).

Since the subject of Quranic language is beyond our scope here, we simply state our stand on it, namely: the Quranic language is unique and stands by itself (as a mu'jizah), hence it must be treated as such. Our task and duty are to rediscover and uncover more of its uniqueness and meaning, through the knowledge extraction process of Quran Analytics.

Linguistics is the scientific study of language. It involves an analysis of language form, language meaning, and language in context, as well as an analysis of the social, cultural, historical, and political factors that influence language.[6] It is a vast field of sciences, with research covering the sub-field of sociolinguistics, neurolinguistics, translations, and many others. The particular field of interest for us is the latest development in computational linguistics. Our current focus will be on the applications of computational linguistics for Quranic studies.

1.4 Computational Linguistics

"Computational linguistics is the scientific and engineering discipline concerned with understanding written and spoken language from a computational perspective, and building artifacts that usefully process and produce language, either in bulk or in a dialogue setting. To the extent that language is a mirror of the mind, a computational understanding of language also provides insight into thinking and intelligence. And since language is our most natural and most versatile means of communication, linguistically competent computers would greatly facilitate our interaction with machines and software of all sorts, and put at our fingertips, in ways that truly meet our needs, the vast textual and other resources of the internet."[7]

The quotes above from the Stanford Encyclopedia of Philosophy reflect the definition of computational linguistics (CL) as a (sub) field of linguistics. The focus of CL is to perform syntactic parsing, word disambiguation, semantic representation and interpretation, making sense of textual representation, and acquisition of knowledge from the language (texts or speeches). The philosophical basis of CL originates from the works of Noam Chomsky, under the generative grammar theory, which claims that grammar in languages follows innate grammatical structures.

The importance of this philosophical basis is to provide scientific justification for us to convert words and sentences, into data, through the filters of grammatical rules. Once this is performed, then we will be able to carry out analytical works using traditional tools of probability and statistics. For this reason, in the early days, the subject is called "statistical linguistics", which today is transformed to "statistical Natural Language Processing (NLP)", or "Natural Language Processing".

1.5 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a combination of linguistics and data science analyzing large amounts of natural language data which includes the collection of speeches, text corpora, and other forms of data generated from the usage of languages. The speeches and text corpora come from official and formal sources

[6] https://en.wikipedia.org/wiki/Linguistics
[7] https://plato.stanford.edu/entries/computational-linguistics/

(e.g., formal speeches, textbooks, etc.) as well as usage in common day-to-day communications (e.g., usage in social media).

NLP has been largely developed over the last few decades and gained large popularity in part due to the development of various algorithms for language processing - from text mining, development of large corpora, all the way to the development of AI applications to language.

The basics in NLP generally cover text parsing and mining, lexical and grammar analysis, semantic analysis. The higher-order works in NLP are pragmatic analysis, socio-linguistics, Machine Learning, and Artificial Intelligence on linguistics - which in turn are used in a variety of applications - from the simplest such as chatbots, all the way to information retrieval and knowledge extraction.

Most text mining is done on ASCII-based alphabets based on UTF-8 Unicode encoding of characters. Particularly, almost all programming languages are based on the English language for its syntax, which drives the easiness of text mining applied to similar characters which are normally or universally used in a computing environment. Despite the availability of UTF-8 for Arabic characters, text mining is just beginning and is still way behind in comparison to the English language. The most important area which is almost relatively unexplored is on cross-language and multi-language text mining, in particular between Arabic and English, Arabic and Malay, Malay and English with particular focus on sacred religious text such as Al-Quran and Al-Hadith.

1.6 Programming language in NLP

Google Trends represents an "unbiased sample" of Google search data, which was made available to the public, representing trillions of searches on the Internet that took place every year. The strength of the tool lies in the availability of historical data, granularized to the details of time (daily) and geographic location (cities) for the whole world.[8] We will use this tool for exploratory views on the usages of computer programming languages in Natural Language Processing (NLP).

Google search for "Top programming language for NLP" reveals that it revolves around five programming languages: MATLAB, R, Python, and Java.[9] Further inspection of trends shows that Python is the most popular by far overtaking all other languages combined.[10] Leading the trend was "Stanford NLP" in Java since Java was used in the Stanford NLP project, "PyTorch" and "spaCy" for Python, since "PyTorch" and "spaCy" are the two most comprehensive and usable libraries for NLP works in Python, and the highest rise of usage is from China. For R, "NLP in R" is the only notable term, reflecting R's lack of a single comprehensive package for NLP.

Stanford CoreNLP, which is among the most comprehensive NLP projects to date and was originally written in Java, now uses Python as its official engine under Stanza. Nevertheless, it can be used by almost all other major programming languages such as PHP, C#, Perl, Ruby, and R.

The latest trends indicate that interest in NLP among the programming languages is very encouraging. Google search for the term "NLP in" is tabulated as follows:[11]

The term	Results
"NLP in R"	43 million
"NLP in Python"	20 million
"NLP in Java"	9.9 million
"NLP in MATLAB"	1.47 million

[8] Google Trends: (https://medium.com/google-news-lab/what-is-google-trends-data-and-what-does-it-mean-b48f07342ee8)
[9] Google search on 24 January 2021
[10] Google trend analysis on 24 January 2021. The search was done on the trends of "web search" on "MATLAB NLP", "R NLP", "Python NLP", and "Java NLP" as the search term for the whole world for the past 5 years.
[11] Search on 24 January 2021

Regular expressions also called *regex*, is an extremely powerful tool for computational linguistics. It is a programming language of its own, which relies on the logic of strings manipulations at the byte level. Regex allows the programmer to manipulate the texts at the rudimentary level, whether to do character search (using byte search) or removal/replacement of characters from text sources based on any structures.

The drawback of regex however is its logic and highly cumbersome codes. Availability of string manipulations libraries, such as "spaCy" in Python or "stringi" in R, makes the use of regex less important in these languages. The basis of the logic applied in most of these libraries originates from regex. This is why regex could prove to be extremely useful in dealing with languages that are not based on Latin-based scripts such as Chinese or Arabic scripts.

The latest trends in programming languages for NLP are encouraging where most solutions are aimed at inter-operability across computing platforms. This is accomplished through the development of Application Programming Interfaces (APIs) and extending the programming language to other languages. For example, TensorFlow which is made famous in Python is extended into R via Keras interface to R. Similarly, many built-in functions which are written in the C++ language are in use by Python, R, and Java. In the future as well as now, the need to be conversant in all programming languages may not be important; as long as one is familiar with any particular popular language, extending it to applications in another language will be straightforward.

1.7 Advancements in NLP and Quran Analytics

Advancements in NLP over the last decade have progressed from simple and basic tasks into Machine Learning, Deep Learning, and Artificial Intelligence with significant success in achieving many complex and complicated operational goals. One of the key areas of success is in transforming data into knowledge. With large amounts of unstructured data, it is extremely difficult to process and convert those data into meaningful representations that humans can derive useful conclusions. What the machine does is to take as input large amounts of data, process them in machine forms using rules generated by NLP tools, and finally present them to humans with the domain knowledge, to interpret them for people with less expertise of the domain concerned.

An example of this is anomaly detection, where given large amounts of data (for example collections of Quranic exegesis), the only way for a human being to detect any "false interpretation" is by reading the texts in detail of all possible collections of exegesis which are considered authentic. The checks must involve all aspects, all possible sources of the interpretation, the methodology used in any specific exegesis, exact context and contents of the interpretation, contextual issues in linguistics, and all relevant notes and pointers attributed by the interpreter. Given the vast amount of exegesis available, what had been done traditionally is to commit everything to memory, and checks are done through memory. Why is this the only possible method? Because even readings on a textual basis will miss small details of anything - such as the exact context of the interpretation, the references used such as other verses, Hadiths, and traditions used. This is almost humanly impossible in this day and age (except for whoever is given the gifts of memorization). Is it possible to use the advancements in NLP to perform this feat? This is an example of applying NLP to a problem in Quranic interpretations.

There are many examples of recent achievements in NLP which are extremely encouraging for a wide range of applications within common usages, such as speech recognition, speech-to-text, information retrieval systems, and others. Since our objective here is to introduce Quran Analytics, we will cover some of the advances particularly towards its potential applications within this field. We have to admit that the coverage on the subject of NLP is very wide and encompasses many domains, therefore our suggestions here remain a preliminary exploration of the subject matter. For further readings on the subject, we recommend that readers refer to the materials suggested at the end of the chapter.

1.7.1 Common NLP tasks

Common NLP tasks are generalized into the following:

1. Properties of natural language - which include phonetics, phonology, syntax, grammar, lexicon, and its variations.
2. Morphology - Part of Speech (POS), lemmatization, and affixations.
3. Semantics - phrase structure, sentences and utterances, relations, thematic, and others.
4. Pragmatics - discourse, intentions, implications, and others.

All the tasks listed above are cast within the linguistic domain, which is based on the notion that language is "generative" in nature. This implies that natural language, whether in the form of speech or textual, is convertible to machine-readable form as data. Once the natural language is converted to data, various algorithms and computing models are applied to generate the tasks in an inference framework. This framework is called "Statistical Natural Language Processing". There are many established references for NLP as a subject, notable among them are Manning and Schutze (1999), Manning et al. (2009), Jurafsky and Martin (2000).

The key aspect of NLP is the statistical analysis, and the tools for it are available in abundance. However, the deeper meaning of the text and inference language is context-dependent. For the English language, this area has been well developed over many decades. Whilst for other languages, it is unclear, in particular the Arabic language, which has its diglossia of colloquial Arabic and Classical Arabic. The Malay language is similar, with its own diglossic dichotomy between Malaysian Malay and Indonesian Malay. All these deeper dimensions may confound statistical inference, and many inferences may suffer from small sample problems.

Lexical analysis is one of the most advanced areas in NLP, due to the digitalization of language as textual data. Lexical analysis relies on two major steps:

a) lexicalization which includes *tokenization*, *lemmatization*, and *word cleaning* process, and
b) analyzing the objects (in this case tokens), within their contexts, such as *syntactic structures*, *grammar analysis*, and others.

Semantic analysis is about relations between structures derived from the data as an outcome of lexical process or engines into the phrases, clauses, sentences, group of sentences as well as the whole text and corpora.

Among the early works for NLP is corpus building. The task is to convert text and generate a large corpus of speech and textual data. There are many notable efforts for public corpus development such as Project Guttenberg[12], American National Corpus[13], British National Corpus[14], BYU corpus linguistics[15], and others. A good list is available on Wikipedia.[16] A paid version program for corpora in many languages is developed by Sketch Engine[17] which has a variety of language capabilities. A sample list for Arabic language corpora is provided by Eddakrouri.[18]

In the past decade, most programming languages have expanded their capabilities to deal with most programming tasks. Furthermore, the works by groups like Stanford NLP, IBM, Google, Baidu, as well as Facebook, generated many language models and treebanks. For example, Stanford NLP through Stanza provided language models for almost all major languages of the world.[19] For treebanks, the famous one is the Penn Treebank project of the University of Pennsylvania, [20] which includes a variety of languages including the Arabic language.[21] Of late, is the development of semantic treebanks, which are collections of natural language annotated with meaning representations. An example of this is the Groningen Meaning Bank[22] by the University of Groningen, Netherlands.

[12]https://www.gutenberg.org
[13]http://www.anc.org
[14]https://www.english-corpora.org/bnc/
[15]https://lcl.byu.edu
[16]https://en.wikipedia.org/wiki/List_of_text_corpora
[17]https://www.sketchengine.eu
[18]https://sites.google.com/a/aucegypt.edu/infoguistics/directory/Corpus-Linguistics/arabic-corpora
[19]https://stanfordnlp.github.io/stanza/available_models.html
[20]https://web.archive.org/web/19970614160127/http://www.cis.upenn.edu/~treebank/
[21]https://catalog.ldc.upenn.edu/LDC2005T20
[22]https://gmb.let.rug.nl

1.7.2 Available resources for digital Quranic studies

Of particular interest for the work in Quran Analytics is the developments in Arabic NLP with a special focus on Classical Arabic. The interest and works in Arabic NLP have only recently started and much more work is still needed. Among the key problems is the availability of standardized Arabic corpora for Modern Standard Arabic.(Marie-Sainte et al., 2019) One of the major limitations for Quran Analytics is the limited availability of resources since Arabic is considered as a "low resource language" in NLP, as compared to "high resource languages" such as English, French, Spanish, German and Chinese.[23] Also, corpora for classical Islamic texts have just begun. A full survey is provided by (Atwell, 2018). The most notable project is by King Saud University Corpus of Classical Arabic, which derived most of its texts from Al-Maktabah Al-Shamelah.[24] It contains Classical Arabic texts from the early stage of Islamic history to modern times.[25]

Since the last decade, many solution providers have developed Quranic studies using web applications or dedicated applications, made available to the public. Notable among them is the Quran corpus project from Leeds University, under *Corpus Quran*.[26] Their effort is highly extensive, since they performed massive efforts in annotating the Quran Arabic from its Uthmani scripts, and created the treebank, ontological graphs, grammatical expressions, as well as its English translations. The web page, as well as the data, has been made as open-source under *Quran Analysis*, [27] and *Text Mining the Quran*.[28]

Quran Gateway [29] is another digital effort, which is provided as a paid version, developed originally by Dr. Andrew Bannister and Dr. Daniel Brubaker. The theme of their works is based on "An Oral-Formulaic Study of the Quran", originally a Ph.D. thesis that was published as a book in (Bannister, 2014). The approach is by taking NLP methods and applying them to the studies of Al-Quran, using both the original Quran in Arabic and various English translations.

Thus far our research reveals that these are the only two works that qualify as comprehensive works on Quranic studies which deploy NLP computing tools for its development. Between the two, the work from the Leeds University working group is more comprehensive and covers wider ranges of applications, and was made to be open source. Quran gateway on the other hand relies on a specific method (or models) which opens it to criticism as narrow-based solutions.

As far as compilations for the translations of Al-Quran, most of the texts (i.e., corpus) are available from www.tanzil.net for various languages.

1.7.3 NLP works on English translations of Al-Quran

Since the most accessible NLP libraries are in the English language, we will focus first on the English translations of Al-Quran. In particular, we will do some simple analytics using two English translations of Al-Quran, namely *The Meaning of the Holy Quran* by Abdullah Yusuf Ali (Yusuf-Ali, 2003) and *The Quran: Arabic Text with Corresponding English Meanings* by Saheeh International (Saheeh-International, 1997). There are other translations such as by A.J. Arberry, Abul Ala Maududi, Pickthall, and a few others. The texts are available for download from www.tanzil.net and other sources. In this book, we will focus mainly on Saheeh and use Yusuf Ali for comparisons. A full work encompassing all other translations is left for the future.

For the English language resources, there are many available models for syntactic, lexical, and treebanks, which are openly accessible. The most notable ones are Stanza NLP from Stanford NLP group, Penn Treebank, as well as Udpipe. We will be using Udpipe because of its easy access within **R**. In the future, we will also use other ready-made libraries for the English language for comparisons.

[23] Hirschberg and Manning (2015)
[24] https://shamela.ws
[25] The site contains over 6,000 texts totaling around 1 billion words. Many of the texts are accessible in https://github.com/OpenArabic/
[26] https://corpus.quran.com
[27] https://www.qurananalysis.com
[28] http://textminingthequran.com
[29] https://info.qurangateway.org/about/

Furthermore, among the libraries for sentiment scoring models, the English language provides many open libraries for usage. Among the notable ones are the "bing sentiment model" (from Bing Liu and collaborators), "AFINN sentiment model" (from Finn Arup Nielsen), "Stanford Sentiment Treebank" (from Stanford NLP group), "nrc sentiment model" (from Saif Mohammad and Peter Turney). We will use bing and AFINN in our work here.

The most notable NLP work on English translation is from the University of Leeds, as indicated earlier. However, the work is just at the beginning stage. Many of their works are available online at www.corpus.quran.com.

1.7.4 Complex NLP tasks for Quran Analytics

In our view, the bigger challenge for Quran Analytics will be in the new advancements of NLP, particularly in the area of information extraction, active learning, augmentation mining, knowledge graphs, knowledge base systems, and probabilistic language modeling. These are among the actively researched areas in NLP.

Information extraction is a technique of extracting structured information from unstructured text. This is a difficult problem due to the complex nature of the human language, whereby words have different meanings when put in different contexts. Active learning on the other hand is important for the tasks of automation in annotation works - which reduces the time for high accuracy annotations tasks. A knowledge graph is a way of storing data that resulted from an information extraction task. It transforms the information data into what is termed a "graph database" concept. Argumentation mining focuses on analyzing argumentation in text and developing tools to automatically extract, aggregate, summarize and reason about arguments in natural language. This in turn provides tools for inference and predictions.

Knowledge base systems (KBS) involve the creation of algorithms that uses knowledge for solving complex problems. An example of this is a set of knowledge settings, such as Islamic rulings (fatwas); a KBS for this purpose is a system of collection of knowledge on all resources relating to the basis of rulings (i.e., Al-Quran, Al-Hadith, etc.). Finally, probabilistic models of language deal with fundamental cognitive questions of how humans structure, process and acquire language. In particular, applications of Deep Learning algorithms and complicated language modeling such as Hidden Markov Models (HMM) are entrenched in this domain.

There are numerous other areas of development within the field of NLP which are not possible to be covered here. What's important is that all these advancements provide massive learning examples which give researchers new and novel methods of exploring NLP further. We are sure that these learning examples are useful tools and resources for Quran Analytics as we envisaged.

1.8 Why use R?

In this section, we will argue for **R** programming language as our tool of choice. **R** is a GNU project, which is a major advantage for many developers and collaborators to join their efforts in developing programs and solutions to common tasks. Only two programming environments have reached a large following in terms of users and contributing collaborators in the field of data science, **Python** and **R**. Which language should you learn? The decision can be challenging because both Python and **R** have clear strengths.[30]

R is exceptional for research – visualizing the data, telling the story, producing reports, and making MVP (minimum viable product) prototype apps with Shiny. From concept (idea) to execution (code), R users tend to be able to accomplish these tasks 3X to 5X faster than Python users, making them very productive for research. Python is exceptional for production – integrating machine learning models into production systems.

R originated as a statistical programming language. This is the reason why it is favored by scientists and researchers more than Python, which is geared more towards programmers. However, it really does not matter which one to choose between Python and **R**, since both are highly interoperable. What is needed is

[30] https://www.r-bloggers.com/2021/02/r-is-for-research-python-is-for-production/

high proficiency in at least one. Most libraries available in Python will have their equivalent in **R**, and vice versa. If it is not available in one, you can simply run the Python program in **R** directly.

We view that in computational linguistics, in particular NLP, **R** is well-positioned to undertake the heavy lifting works required. In particular, the flexibility of **R** as a Massively Objected Oriented Programming (MOOP) language with its advanced data structures, gives us flexibilities to construct our own data structures fitting the problem at hand. This will become handy when we have to deal with complex concepts and methods for the Quranic language, in particular the Quranic Arabic. As far as the Arabic scripts are concerned, they can be dealt with quite easily; and usage of regular expressions is extremely straightforward in **R**.

More importantly, developers have built massive libraries (called packages) in **R**, from a very broad discipline of subjects and applications. As of today, there are more than 10,000 packages that have been developed and maintained by CRAN, which is the common **R** codes repository, available for download and usage. Of course, this is a far cry from Python, which has more than 200,000 packages, hosted by PyPI. The advantage of **R** and *CRAN* is that all packages in the repository are well maintained, versioned, and well documented. In **R**, the most popular IDE (Integrated Development Environment) is *R Studio*, which is easy to use, and all documents are designed for reproducibility through *R Markdown*.

Lastly, we would argue that as far as NLP is concerned, **R** represents the most vibrant alternative to Python, and is used widely by many NLP researchers and developers as a tool of choice. This is evident from the search results in the previous section which shows that interest in "NLP in R" is almost double the interest in "NLP in Python". Since **R** is more attuned to researchers and scientists, we will take R as our programming language of choice. For a good introduction to **R**, we recommend "R for Data Science" by Wickham and Grolemund (Wickham and Grolemund, 2016), its online version is also available,[31] and "R Programming for Data Science" by Roger Peng (Peng, 2014), with its online counterpart.[32]. For a good course on R is from Coursera.[33]. A full exposition of the R programming language is "Extending R" by (Chambers, 2016).

1.9 Focus of this book

The purpose of this book is to guide our readers along with the exploratory data analysis (EDA) step in data science.[34]. The datasets we will explore are mainly the selected English translations of Al-Quran. The methods of the exploratory analysis will be the various tools available, mainly using **R** programming language. This is intended as a first step before we will apply it within a larger context of applications on the original language of Al-Quran, which is in the Uthmani script and based on Al-Quran Classical Arabic, relying on various classical Islamic texts, such as the Hadith, Tafseer, and others which are based on Classical Arabic.

We intend to achieve three indirect objectives in this book:

1. To indirectly introduce R programming language as a data science tool.
2. To introduce some of the NLP tools in R.
3. To explore some of the NLP concepts for text and data analysis.

The more direct objectives of our exploratory work are to:

1. Demonstrate the usage of various tools within the context of the English language for the Quran English translations, and provide indications in terms of applications within the original text of Al-Quran.
2. Suggest various paths of research and development that can be further expanded based on the lessons learned from dealing with the English language. These are noted at the end of each chapter as conclusions providing further direction of research.

[31] https://r4ds.had.co.nz
[32] https://bookdown.org/rdpeng/rprogdatascience/
[33] https://www.coursera.org/specializations/data-science-foundations-r
[34] https://r4ds.had.co.nz/

3. Showcase various existing open-source models of language, in particular the English language. This will be shown through various comparative analyses.

Most importantly, we hope that our work will spur more interest among researchers and collaborators to further investigate and expand the area of Quran Analytics, Natural Language Processing, Linguistics, in totality and comprehensively. It should encompass the domain of English language (for the case of English-speaking people), Malay language (for southeast Asian Malays), and hopefully other languages such as Farsi, Turkish, Chinese, Bangladeshi, Hindi, and numerous other major languages that have their Quran translations.

1.10 Further readings

Atwell, Eric. *Classical and Modern Arabic Corpora: Genre and Language Change.* John Benjamins Publishing Company, Amsterdam, The Netherlands, 2018. (Atwell, 2018)

Baayen, Harald R. *Analyzing Linguistic Data: A Practical Introduction to Statistics Using R.* Cambridge University Press, Cambridge, United Kingdom, 2008. (Baayen, 2008)

Chambers John M. *Extending R.* Chapman Hall/CRC, Boca Raton, Florida, 2016 (Chambers, 2016)

Gries, Stefan T. *Quantitative Corpus Linguistics with R: A Practical Introduction.* Routledge, Taylor & Francis Group, New York, New York, 2017. (Gries, 2017)

Izutsu, Toshiko. *God and Man in the Qur'an.* Keio University, Tokyo, Japan, 1964 (Izutsu, 1964)

Jurafsky, Daniel and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice Hall, New Jersey, United States, 2000 (Jurafsky and Martin, 2000)

Manning, Christopher D. and Hinrich Schutze. *Foundations of Statistical Natural Language Processing.* The MIT Press, Cambridge, Massachusetts, 1999 (Manning and Schutze, 1999)

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schutze. *An Introduction to Information Retrieval.* Cambridge University Press, Cambridge, England, 2009. (Manning et al., 2009)

Peng, Roger D. *R Programming for Data Science.* Leanpub, Victoria, British Columbia, Canada, 2014. (Peng, 2014)

Rahman, Fazlur. *Major Themes of the Qur'an.* The University of Chicago Press, Chicago, Illinois, USA, 1980. (Rahman, 1980)

Saeh, Bassam. *The Miraculous Language of Qur'an: Evidence of Divine Origin.* International Institute of Islamic Thought, London, United Kingdom, 2015 (Saeh, 2015)

Von-Denfer, Ahmad. *Ulum Al-Quran: An Introduction to the Sciences of the Quran.* The Islamic Foundation, London, United Kingdom, 1983 (Von-Denfer, 1983)

Wickham, Hadley and Garret Grolemund. *R for Data Science.* O'Reilly Media, Inc, Newton, Massachusetts, 2016. (Wickham and Grolemund, 2016)

Analysis of Words by its Frequencies

> "What are the common patterns that occur in language use?" The major tool which we use to identify these patterns is counting things, otherwise known as statistics" (Manning and Schutze, 1999, pg. 4)

The most basic element of text corpora is the word. Words form sentences. Sentences are organized as groups of words. A corpus consists of sentences assembled together as paragraphs, sections, chapters as a set of documents. The elementary first step of Quran Analytics is to deal with the arrangement of words in the corpus. This is to reveal its basic structure, frequencies, in which part of the corpus the word appears, and any other particular observation of interest. This exercise is a generic statistical analysis in NLP.

Al-Quran consists of 77,430 words, 6,236 verses, and 114 Surahs - arranged in a specific order from the first word and verse, to the last verse and word. The verses differ in length as well as the Surahs. Any translation of Al-Quran follows a similar structure, except for the words in each verse that may differ, based on the language and style of the authors.

Due to this unique arrangement, it is important to investigate the structures of the translations of Al-Quran (i.e., a corpus), in any language, sourced from the same original texts. These structures reveal the linguistic styles and the messages (or knowledge) transmission process of the texts.

This part consists of two chapters, Chapter 2 and Chapter 3. In Chapter 2 we will cover the statistical analysis of translations of Al-Quran. In Chapter 3 we choose one of the methods to deal with words by applying selected word scoring models, namely sentiment scoring. A full exercise for both tasks requires lengthy expositions, which is not our intention. Instead, we provide a sketch of ideas of possible works as a guide for full-scale analysis.

We will utilize **R** programming language, introduce some of the packages, and utilize some of the functions within these packages. We attach some of the codes wherever relevant and focus on visualizations of the results.

2 Word Frequency Analysis

The first task in word analytics in a corpus is to study the words' statistical properties. Words represent the building block of sentences, which form the building block of a corpus, a collection of texts. Words are the primary form of "data" and hence convertible to frequencies. The most straightforward analysis for frequencies will be through analyzing its statistical properties.

Before we perform the analysis, we must first convert the words into **tokens**. In the process, we have to clean up all non-words in the texts, such as punctuations, symbols, and other characters, and convert all texts into lowercase. After that, we convert all the texts into tokens and create the annotations for those tokens. The annotations are markers such as a sentence, which part of a sentence, and which corpus.

In **R**, we can use many ready-made packages, such as *tm* (Feinerer et al., 2020) (stands for text mining) and *tidytext* (Queiroz et al., 2020), to perform the annotation works. *tidytext*'s advantage is that it automatically performs all the cleaning works needed and formats them into *tidy data*. For this chapter, we will use *tidytext* as the package of choice. For reference on *tidytext*, please refer to *Text Mining with R: A Tidy Approach* (Silge and Robinson, 2017), and for a comprehensive introduction to the word statistical analysis, please refer to *Foundations of Statistical Natural Language Processing* (Manning and Schutze, 1999). For plotting, we will use *ggplot2* (Wickham et al., 2020) package throughout the whole book as our standard of "grammar of graphics".

The objective of our work in this chapter is to introduce some of the statistical analysis on words. In particular, we want to analyze two versions of the English translation of Al-Quran, namely *Quran Saheeh International* (Saheeh-International, 1997), *The Meaning of the Holy Quran by Abdullah Yusuf Ali* (Yusuf-Ali, 2003), and a Malay translation obtained from https://www.surah.my. Saheeh is published in 1997 by Dar Abul Qasim, Saudi Arabia, translations by three American ladies using what is termed as "un-archaic" language. On the other hand, Yusuf Ali is published for the first time in 1937 from the work of Abdullah Yusuf Ali, who is a Shia in the Dawoodi Bohra tradition[35] using British English of the time. The Malay translation originates from *Tafseer Pimpinan Ar-Rahman* by Abdullah Basmieh, a well-known and widely accepted Al-Quran translation for the Malay language.

[35] https://en.wikipedia.org/wiki/Abdullah_Yusuf_Ali

Since all translations are from the same source of Al-Quran in Arabic, based on different times and styles (for English) and in another separate language (for Malay), it is interesting to study them using word statistical analysis. We may ask the questions, are these translations, in terms of words, different or similar, from a word statistical analysis point of view. These are the types of analysis we intend to accomplish in this chapter.

2.1 R packages and data used

We will use the *tidytext* package in **R**. For the data, we will use the data from the *quRan* package developed by (Heiss, 2018), which contains four sets of data, namely: *quran_ar* (Quran Arabic), *quran_ar_min)* (Quran Arabic minimized), *quran_en_sahih* (Sahih), and *quran_en_yusufali* (Yusuf Ali). We will use our dataset for the Malay translation which we generate directly from the source through a parser. We combined all the datasets into a single dataset, called *quran_trans.csv*.

```
library(tidyverse)
library(tidytext)
library(readr)
library(quRan)
library(ggplot2)

quran_all = read_csv("data/quran_trans.csv") %>%
              select(surah_id, ayah_id, surah_title_en,
                     surah_title_en_trans, revelation_type,
                     ayah_title,
                     malay,saheeh,yusufali)
```

To work with the data as a tidy dataset, we need to restructure it in the one-token-per-row format. We will use the *unnest_tokens()* function from the *tidytext* package. Tokenization is the first step in word analytics.

```
tidyESI <- quran_all %>%
  unnest_tokens(word, saheeh) %>% select(-malay,-yusufali)
tidyEYA <- quran_all %>%
  unnest_tokens(word, yusufali) %>% select(-malay,-saheeh)
tidyMAB <- quran_all %>%
  unnest_tokens(word, malay) %>% select(-saheeh,-yusufali)
```

The *unnest_tokens* function uses the *tokenizers* package to separate each line of text in the original data frame into tokens. The default tokenizing is for *words*, but other options include *characters*, *n-grams*, *sentences*, *lines*, *paragraphs*, or separation around a regex pattern. The total number of tokens represents the total number of words in the whole corpus, from the first to last. After tokenization, we can calculate for each corpus the number of tokens present: 158,065 tokens for Saheeh, 167,859 for Yusuf Ali, and 204,784 for Malay. The comparable number for the original Al-Quran Arabic is 77,430 words (tokens)[36]. Overall comparison with the Arabic texts, clearly indicates that Saheeh, Yusuf Ali (the English corpora), and Malay (the Malay corpus) are much more verbose (more than double for the English, and almost triple for the Malay).

We can see that for Saheeh and Yusuf Ali, while both translate the same original Quran, the number of total words differs by 9,794. This is an indication that Yusuf Ali is more verbose than Sahih by 6.2 percent. Why is Yusuf Ali much more verbose? Probably the style of the English language and method of translation is different between the two. In contrast, the total number of words for Malay exceed Saheeh by 46,719, or about 30 percent more verbose than the Saheeh's English.[37]

[36] Dukes and Habash (2010)

[37] This may indicate that in general, the Malay language is more verbose than the English language or the author may have included some commentaries within the text of the verse, usually in parenthesis.

2.2 Wordclouds analysis

Wordcloud analysis is a simple visual representation of word occurrence frequency within a set of text. A wordcloud visual is a tool to identify keywords, whereby a larger image shows a more common word. It is extremely useful for analyzing a large body of texts, especially unstructured text data. It is commonly used when we have to analyze large amounts of texts in big data applications.

In **R**, there are many packages useful for wordcloud analysis, the easiest one is the *wordcloud* package. We will show how to use them here.

```
library(wordcloud)
library(RColorBrewer)
set.seed(1234)
tidyESI %>%
  count(word) %>%
  with(wordcloud(words = word,
                 freq = n,
                 max.words = 200,
                 random.order = FALSE,
                 rot.per = 0.35,
                 colors=brewer.pal(8,"Dark2")))
```

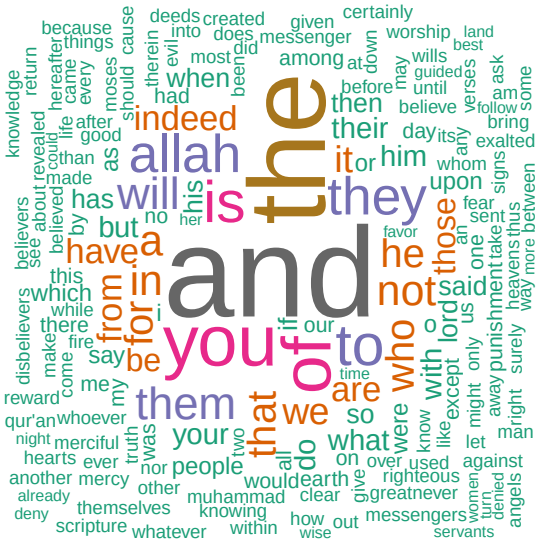


Figure 2: Wordcloud for Saheeh translation

For Saheeh, the wordcloud plot is in Figure 2; for Yusuf Ali, the wordcloud plot is in Figure 3; and for the Malay, the wordcloud plot is in Figure 4.

Obviously, the words such as "the" and "and" dominate the English versions, and "yang" and "dan" dominate the Malay version. These are "stopwords" in the languages, which we will deal with later. An English linguist may be able to discern the styles between Saheeh (which is American English) and Yusuf Ali (which is slightly older Engish) by observing the dominant usages of these stopwords.

Wordcloud tool is an easy and convenient way of visualizing word frequencies which we will be using throughout the book.

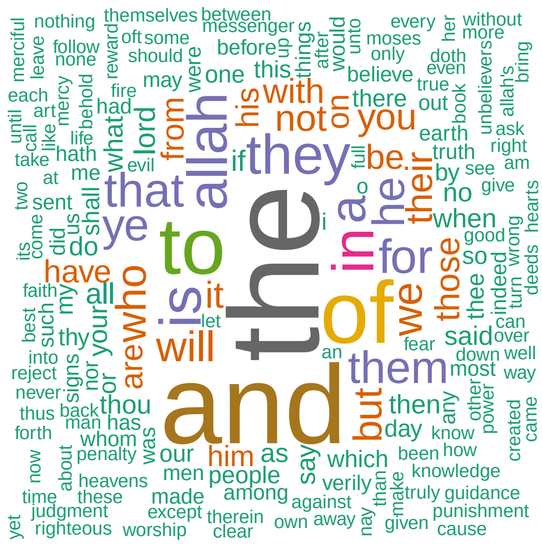


Figure 3: Wordcloud for Yusuf Ali translation

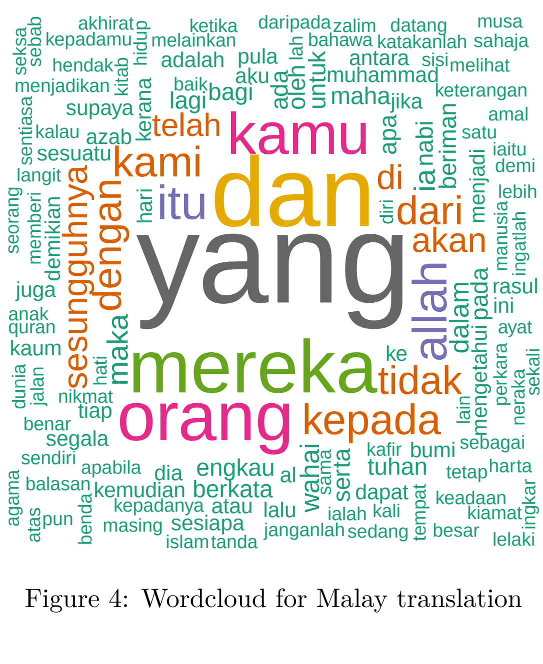


Figure 4: Wordcloud for Malay translation

2.3 Analyzing word and document frequency

A central question in text mining and Natural Language Processing (NLP) is how to quantify what a document is all about. We can do this by looking at the words that make up the document. One measure of the importance of any word is its term frequency (tf), how frequently a word occurs in a document. There are words in a document that occur many times but may not be of any importance; in English, these are probably words like "the", "is", "of", and so forth; in Malay, these are words like "dan" and "yang". We might take the approach of adding words like these to a list of stopwords and removing them before analysis, but some of these words might be more important in some documents than others. A list of stopwords is not a very sophisticated approach for adjusting the term frequency for commonly used words.

Another approach is to look at a term's inverse document frequency (idf), which decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents. We can use the term frequency to calculate its tf-idf (the two quantities multiplied together), a measure of the term's frequency adjusted for how rarely it occurs in the whole text.

The tf-idf statistic is a measure of how important a word is to a document in a collection (or corpus) of documents, for example, to one novel in a collection of novels or one website in a collection of websites. In the case of the Quran, we can compare its properties between Surahs or Juz or Hizb.

The tf-idf is a rule-of-thumb or heuristic quantity. Simultaneously, it has proved useful in text mining and search engines. Its theoretical foundations are considered less than firm by information theory experts. The inverse document frequency for any given term is defined as:

$$idf(term) = \frac{ln[(numberofdocuments)]}{(numberofdocumentscontainingterm)}$$

and tf-idf is defined as:

$$tfidf(term) = tf(term) * idf(term)$$

We can use tidy data principles to approach tf-idf analysis and use consistent, practical tools to quantify how various important terms are in a document that is part of a collection.

2.3.1 Term frequency in English Quran

We start by looking at the Surahs of the Quran and examine first term frequency, and then tf-idf. We can start just by using *dplyr* verbs such as *group_by()* and *join()*. We also calculate the total words in each Surah here, for later use.

```
surah_wordsESI <- quran_all %>%
  unnest_tokens(word, saheeh) %>%
  count(surah_title_en, word, sort = TRUE)
surah_wordsEYA <- quran_all %>%
  unnest_tokens(word, yusufali) %>%
  count(surah_title_en, word, sort = TRUE)
surah_wordsMAB <- quran_all %>%
  unnest_tokens(word, malay) %>%
  count(surah_title_en, word, sort = TRUE)

total_wordsESI <- surah_wordsESI %>%
  group_by(surah_title_en) %>% summarize(total = sum(n))
total_wordsEYA <- surah_wordsEYA %>%
  group_by(surah_title_en) %>% summarize(total = sum(n))
total_wordsMAB <- surah_wordsMAB %>%
  group_by(surah_title_en) %>% summarize(total = sum(n))

surah_wordsESI <- left_join(surah_wordsESI, total_wordsESI)
```

```
surah_wordsEYA <- left_join(surah_wordsEYA, total_wordsEYA)
surah_wordsMAB <- left_join(surah_wordsMAB, total_wordsMAB)
```

In the above codes, we create *data.frame surah_wordsESI*, *surah_wordsEYA*, and *surah_wordsMAB*, for each word-surah combination; n is the number of times that word is used in the Surah and *total* is the total words in the Surah. The usual suspects with the highest n are "the", "and", "to" (for English), and "dan", "maka" (for Malay).

In the following figures, we look at the distribution of $\frac{n}{total}$ for surahs grouped by the Surahs' length. The term frequency is the number of times a word appears in a surah divided by the total number of terms (words) in that Surah.

We start first with the short Surahs at the end of Al-Quran, then to the medium length Surahs, and finally to the long Surahs (similar to when we normally start to learn Al-Quran, starting to learn reading Al-Quran by learning the shorter ones first, followed by the longer ones).

```
last6_surahs = c("An-Naas", "Al-Falaq", "Al-Ikhlaas", "Al-Masad","An-Nasr", "Al-Kaafiroon")
hamim_surahs = c("Ghafir", "Fussilat", "Ash-Shura", "Az-Zukhruf", "Ad-Dukhaan", "Al-Jaathiya")
long_surahs = c("Al-Baqara", "Aal-i-Imraan", "An-Nisaa", "Al-Maaida","Al-An'aam", "Al-A'raaf")
```

```
tf_plotter = function(df,surah_group,title_label,x_lim){
  df %>%
  filter(surah_title_en %in% surah_group) %>%
  ggplot(aes(n/total, fill = surah_title_en)) +
  geom_histogram(show.legend = FALSE) +
  xlim(NA,x_lim) +
  facet_wrap(~surah_title_en, ncol = 2, scales = "free_y") +
  labs(title = title_label,
       x = "term frequency",
       y = NULL)
}
```

From these plots of "term frequency" starting with the short Surahs (Figure 5, Figure 6, and Figure 7), followed by the medium Surahs (Figure 8, Figure 9, and Figure 10), and the long Surahs (Figure 11, Figure 12, and Figure 13), there are many insights and lessons about the language of the translation of Al-Quran.

For example, we can observe that the term frequencies for all the translations showed "scale-invariant properties" to the language and terminologies used to translate the Quran. Scale-invariant refers to the fact that despite different filters used (a translation is a filtering method), the resulting outputs exhibit similar properties; this is an important observation about the structure of the original texts of the Quran in Arabic. An example of scale-invariancy is the sign languages for deft and mute people - whereby it is known to be language-specific independent. Are the translated messages in the Quran, aside from the original text, language-independent?

There are many more analyses open to the researchers by studying the various structures and properties of the word frequencies in the original texts against the translated texts of Al-Quran. We leave this for future work and research.

To have a broader view of the observations made, let us look at the plot for the percentages of unique words (tokens) against the total number of words (tokens) within each Surah for all the translations.

```
library(reshape2)
EQT_total_toks = data.frame("Surah" = 1:nrow(ESI_total_toks),
                            "Saheeh" = ESI_total_toks$percentage,
                            "YusufAli" = EYA_total_toks$percentage,
                            "Malay" = MAB_total_toks$percentage)
```

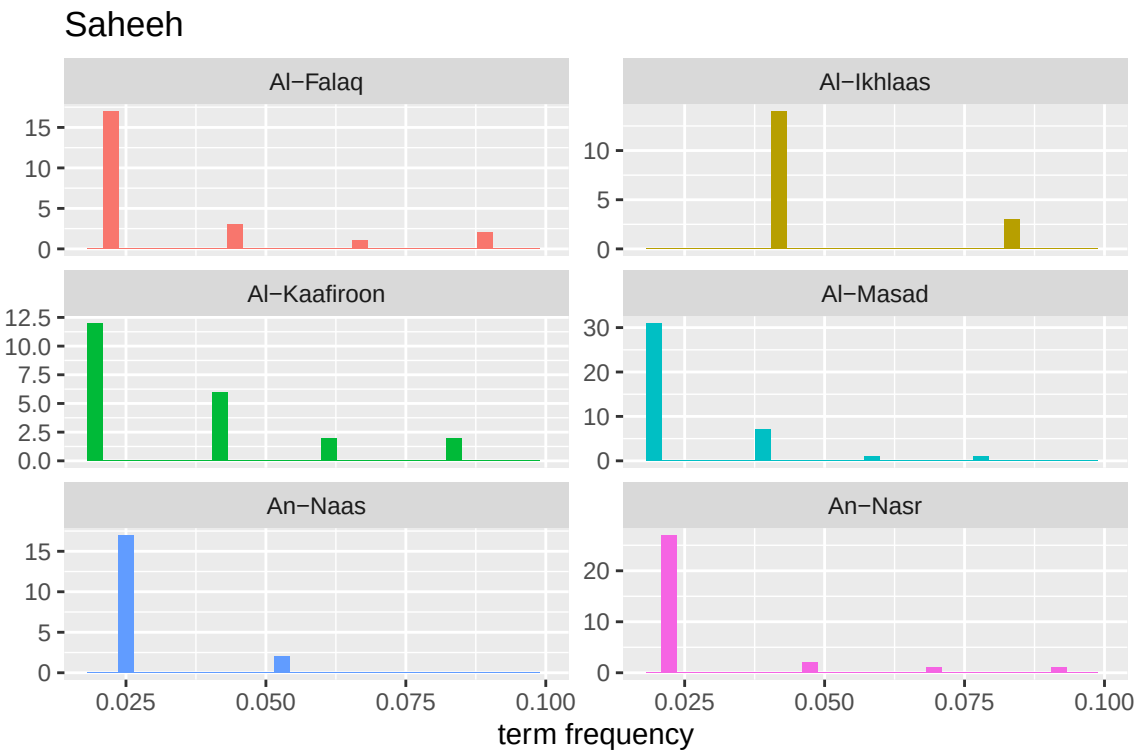


Figure 5: Short Surahs term frequency in Saheeh

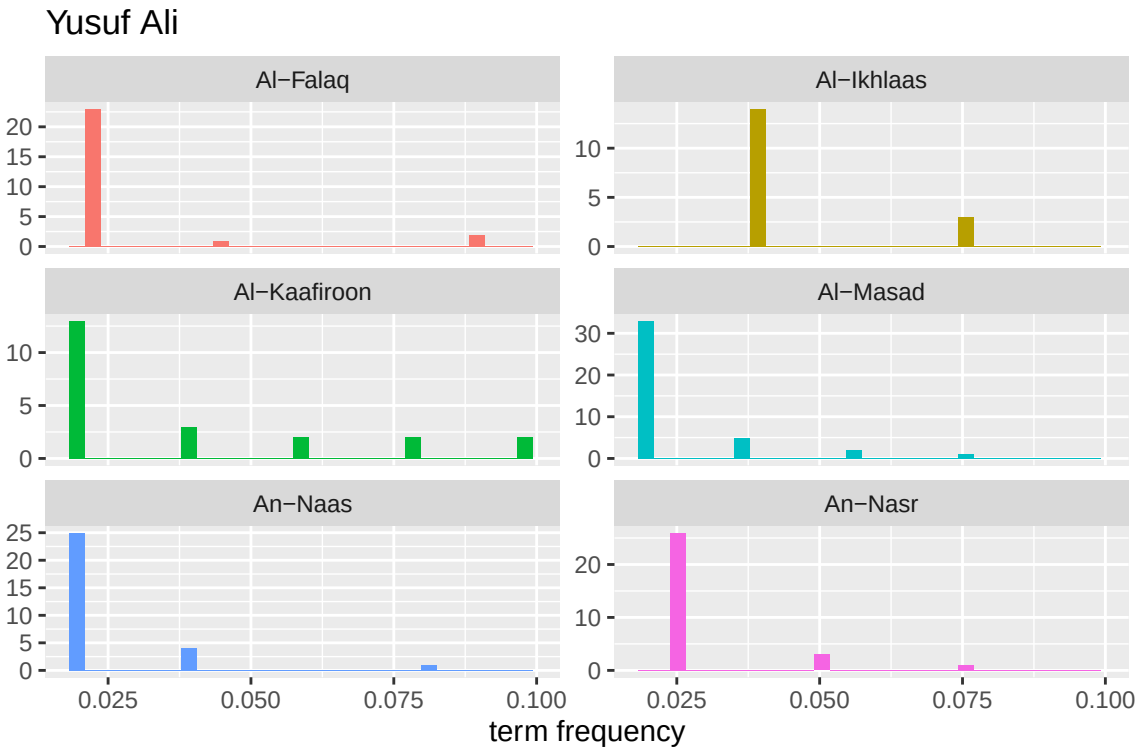


Figure 6: Short Surahs term frequency in Yusuf Ali

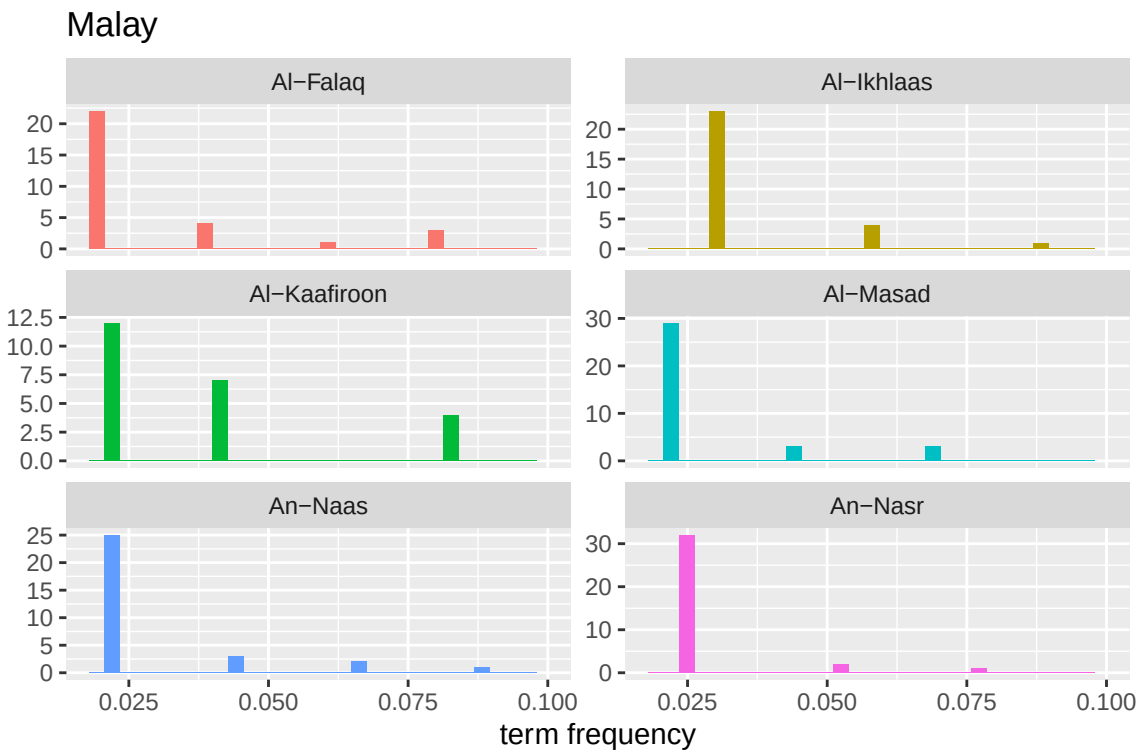


Figure 7: Short Surahs term frequency in Malay

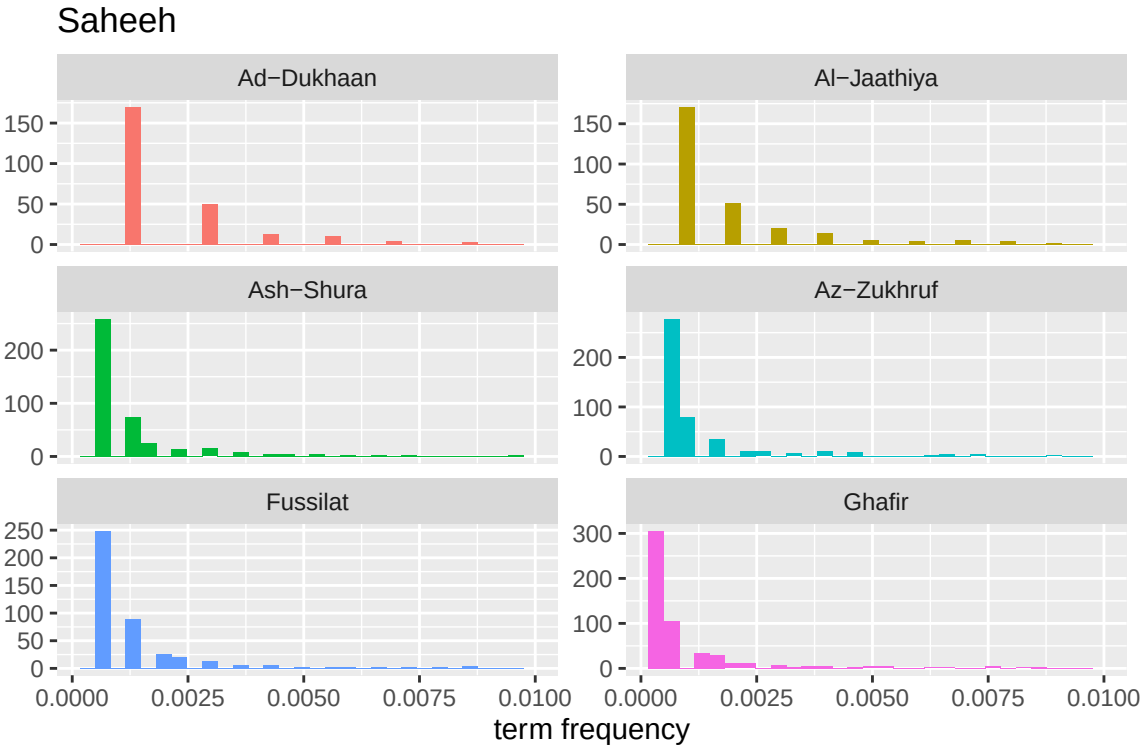


Figure 8: Hamim Surahs term frequency in Saheeh

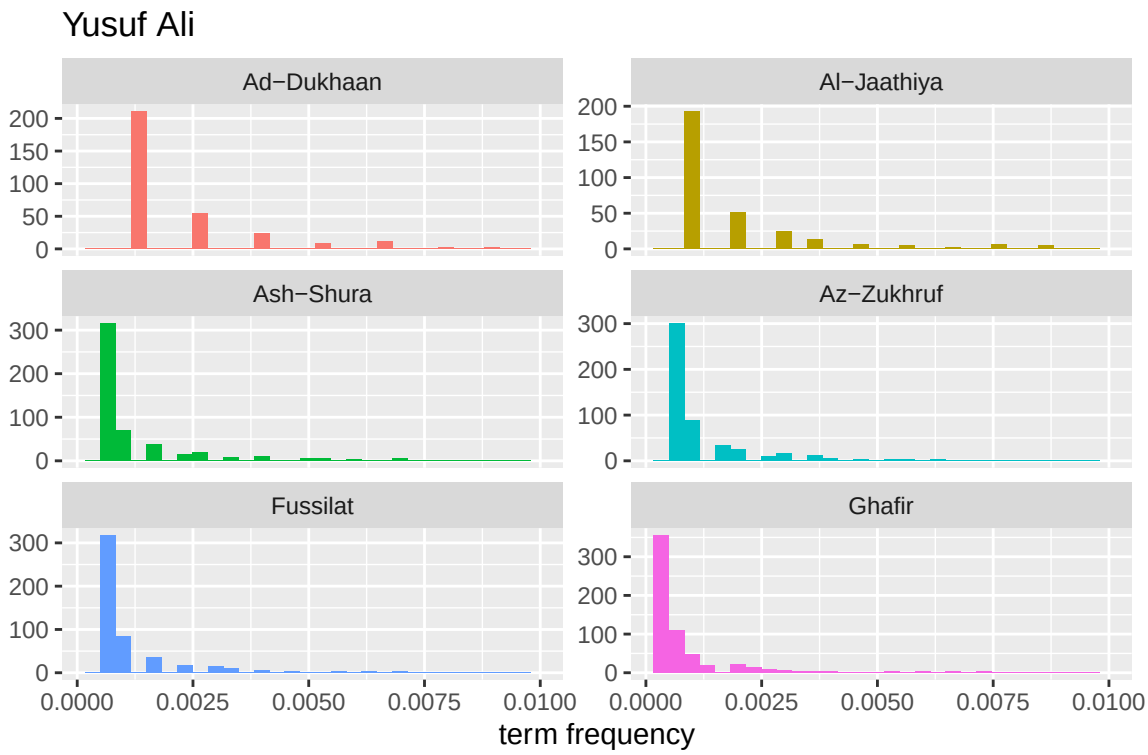


Figure 9: Hamim Surahs term frequency in Yusuf Ali

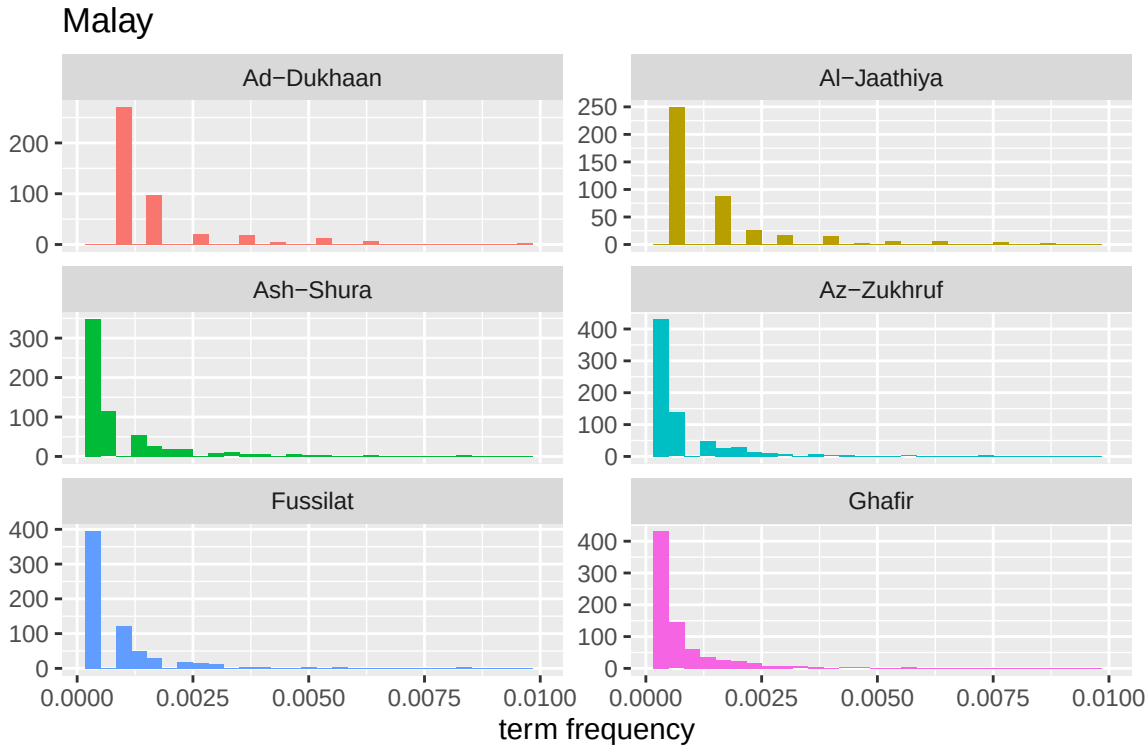


Figure 10: Hamim Surahs term frequency in Malay

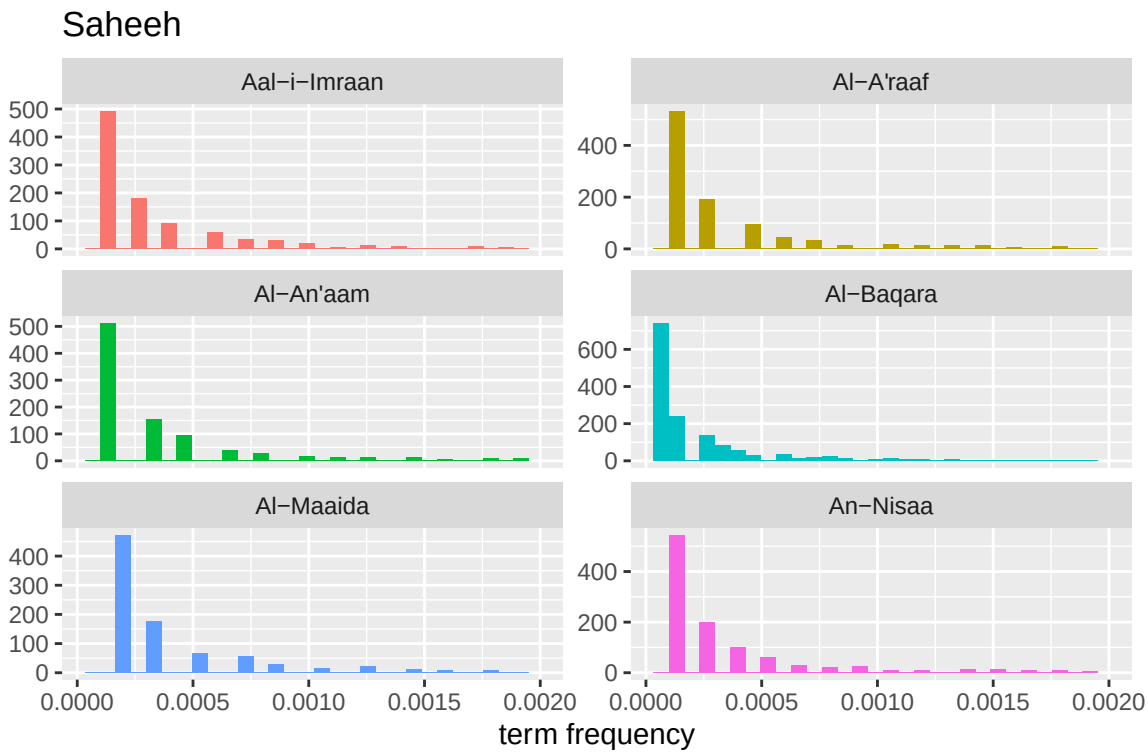


Figure 11: Long Surahs term frequency in Saheeh

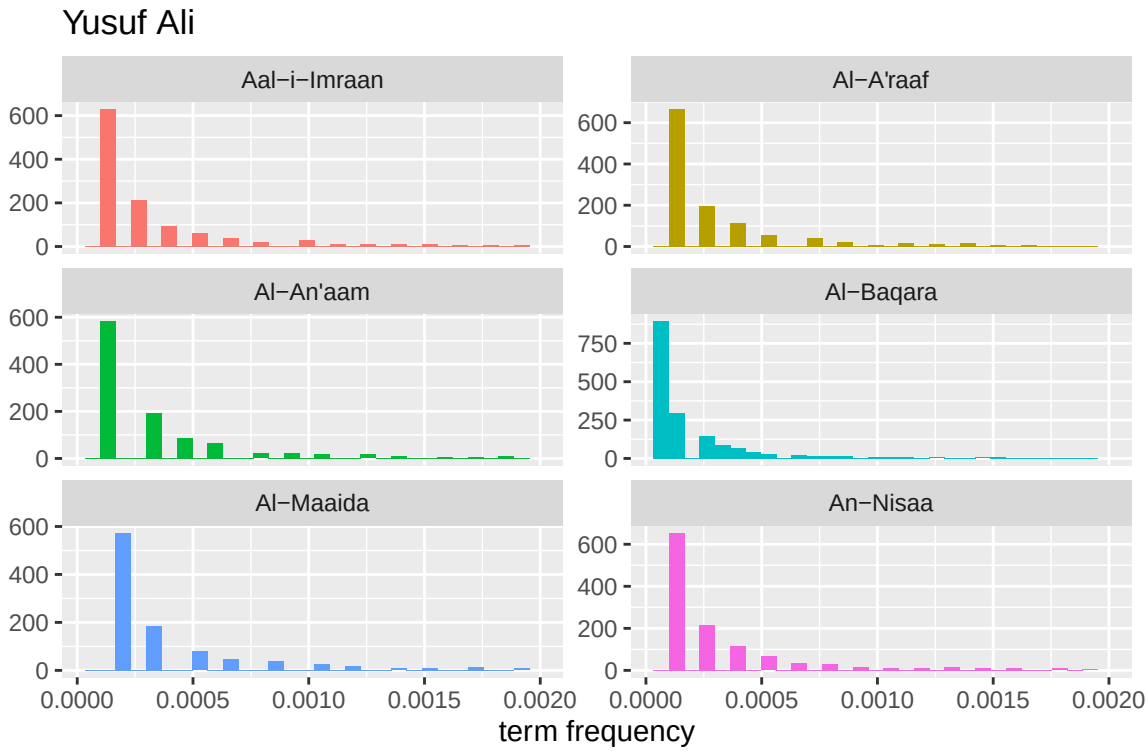


Figure 12: Long Surahs term frequency in Yusuf Ali

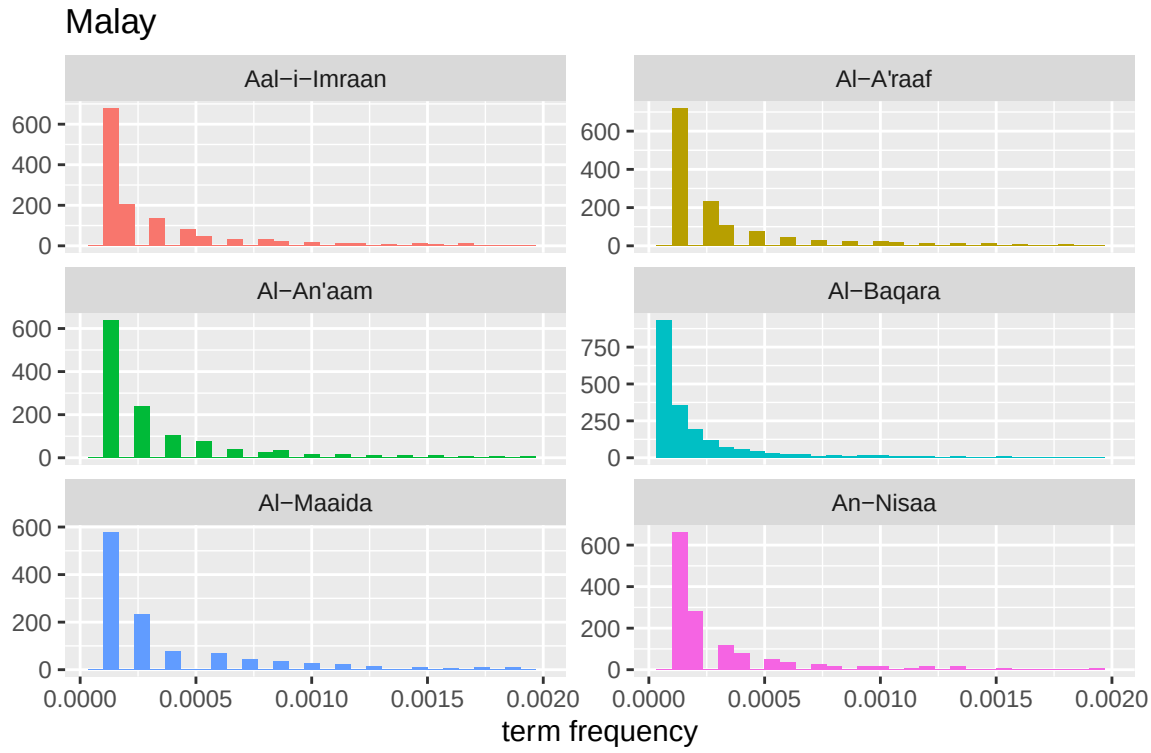


Figure 13: Long Surahs term frequency in Malay

```
EQT_total_toks_melt = melt(EQT_total_toks,
                           id.vars = "Surah",
                           variable.name = "Translation",
                           value.name = "Percentage")
EQT_total_toks_melt %>% ggplot() +
                            geom_point(aes(x = Surah,
                                           y = Percentage,
                                           shape = Translation,
                                           color = Translation))  +
                            labs(x = "Surah number",
                                 y = "Percentage")
```

The percentage of "unique words" over "total words" is a measure of the lexical variety within a sub-set of texts (i.e., Surah). The plot in Figure 14 shows that the lexical variety in the shorter Surahs varies more than the longer Surahs. It is quite rare in any regular text collection that a shorter group of sentences (such as chapters) display more outstanding lexical varieties. The case is different here.

We can see that while all the translations differ in the languages and styles, the lexical varieties in the shorter Surahs are consistently higher. An implication of this is that, while many of the Surahs in Al-Quran may be short, they convey distinctively different messages.

2.3.2 The *bind_tf_idf* function

The concept of tf-idf is to measure the degree of importance of words within the content of each group of texts, such as a Surah, by decreasing the weight for commonly used words and increasing the weight for words used less. We want to detect commonly occurring words, but not too common. In general textual

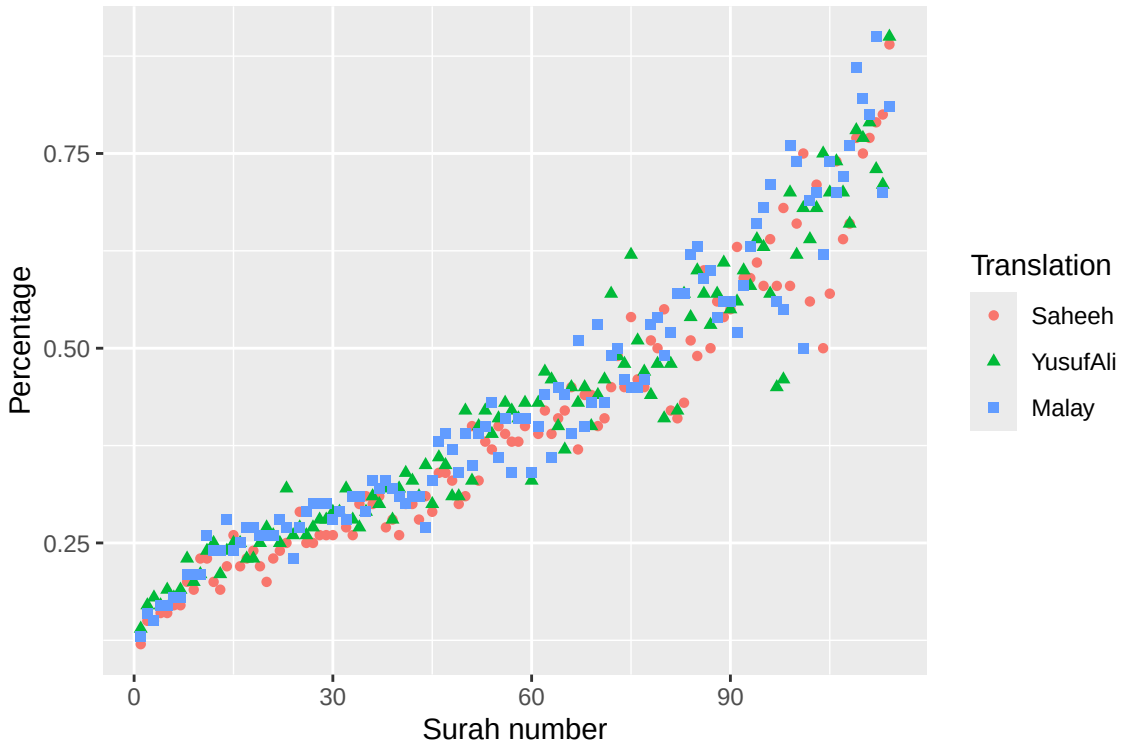


Figure 14: Percentage of unique words in the Surahs

analysis, these words represent a topic of interest, the headlines, the themes, or a subject that stands above other surrounding texts. It is an essential tool for understanding text structures and uncovering the messages in texts.

The idf's and thus tf-idf's for the highly used words, such as the word "the", is zero. These are all words that appear in all 114 Quran Surahs, so the idf term (which will then be the natural log of 1) is zero. It is also very low (near zero) for words that occur in many documents (i.e., Surahs) in a corpus. Furthermore, the inverse document frequency will be higher for words that occur in fewer of the sub-set of documents (i.e., Surahs) in the corpus. Therefore, words of which idf's and tf-idf's are near zero, and yet not near enough are what we are looking for.

The *bind_tf_idf* function in the *tidytext* package takes a *tidytext* dataset as input with one row per token (term), per document. One column (a column named *word*) contains the terms/tokens, one column contains the documents (Surah in this case), and the last necessary column contains the counts, how many times each document contains each term (n in this example).

First, let us plot the tf-idf for all three translations and make some observations.

```
surah_wordsESI <- surah_wordsESI %>%
  bind_tf_idf(word, surah_title_en, n)
surah_wordsEYA <- surah_wordsEYA %>%
  bind_tf_idf(word, surah_title_en, n)
surah_wordsMAB <- surah_wordsMAB %>%
  bind_tf_idf(word, surah_title_en, n)
ESI_tf_idf = surah_wordsESI %>%
                select(-total) %>% arrange(desc(tf_idf))
EYA_tf_idf = surah_wordsEYA %>%
                select(-total) %>% arrange(desc(tf_idf))
MAB_tf_idf = surah_wordsMAB %>%
                select(-total) %>% arrange(desc(tf_idf))

tfidf_plotter = function(df_plot,title_label,color){
```

```
        ggplot() +
        geom_point(aes(x = 1:length(df_plot$tf_idf),
               y = log(df_plot$tf_idf)), color = "red",
               size = 0.05) +
        labs(title = title_label, x = "n", y = "tf_idf")}
```

Figure 15: tf-idf for the translations

We can see from Figure 15, the structure for all the translations looks similar; the differences are due to the number of words in each translation. What is striking is despite all the variety of words and language used, the structure of tf_idf is almost perfectly the same (and in fact, they are the same if we normalize the scale by the number of total words). It is clear that there are some words which used rarely (the curves on the left side, with high tf_idf), the numbers of which are not that many, and there are words that are used extremely frequently (the curves on the right side, with low tf_idf), the numbers of which are not that many; and the rest of the words are used moderately (in between).

Let us look at a visualization for these high tf-idf words. These are shown in Figure 16, Figure 17, Figure 18, Figure 19, Figure 20, Figure 21, Figure 22, Figure 23, and Figure 24.

As measured by the tf-idf, these words are the most important to each Surah, and most readers would likely agree. It identifies words that are important to one document within a collection of documents. Furthermore, we can see that Saheeh's translation used different words than Yusuf Ali; this is obvious in the top tf_idf terms where they differ significantly.

The shape of the tf-idf curves has many similarities between the translations; the terms (or words) they use are not the same, while clearly, they are only translations of the same Arabic verses. These observations raise a question: do different translation methods bring different meanings to the readers? Will this alter the message contained in the original texts?

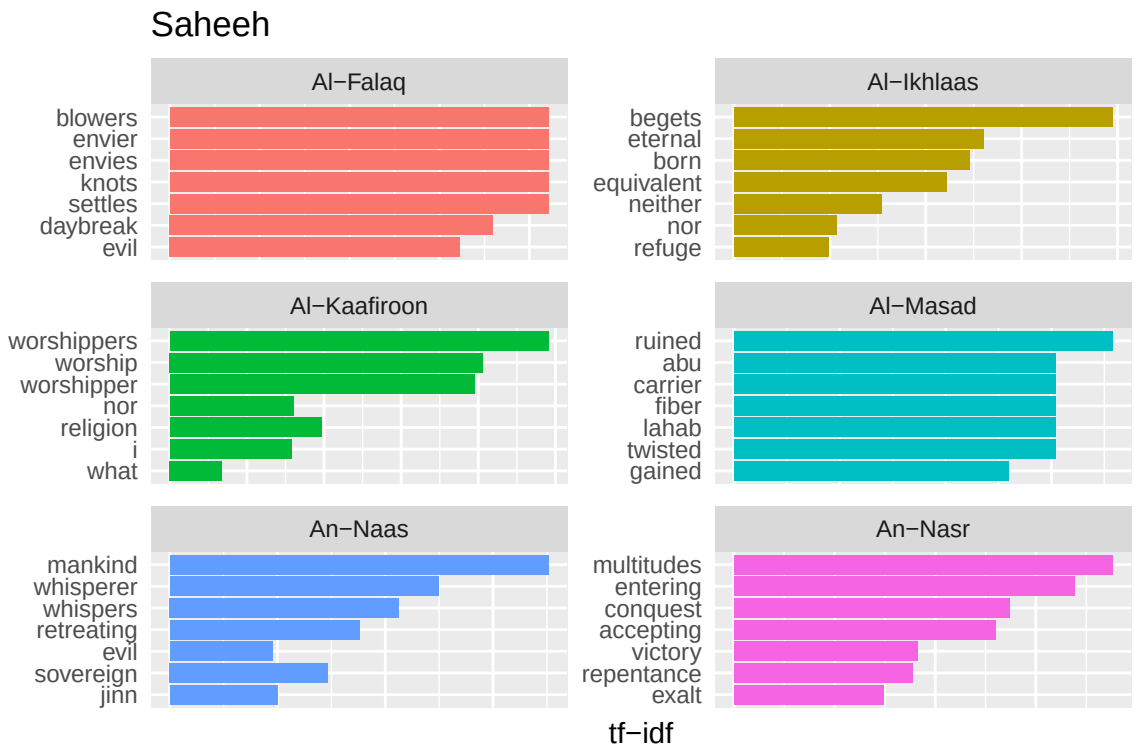


Figure 16: Short Surahs tf-idf for Saheeh

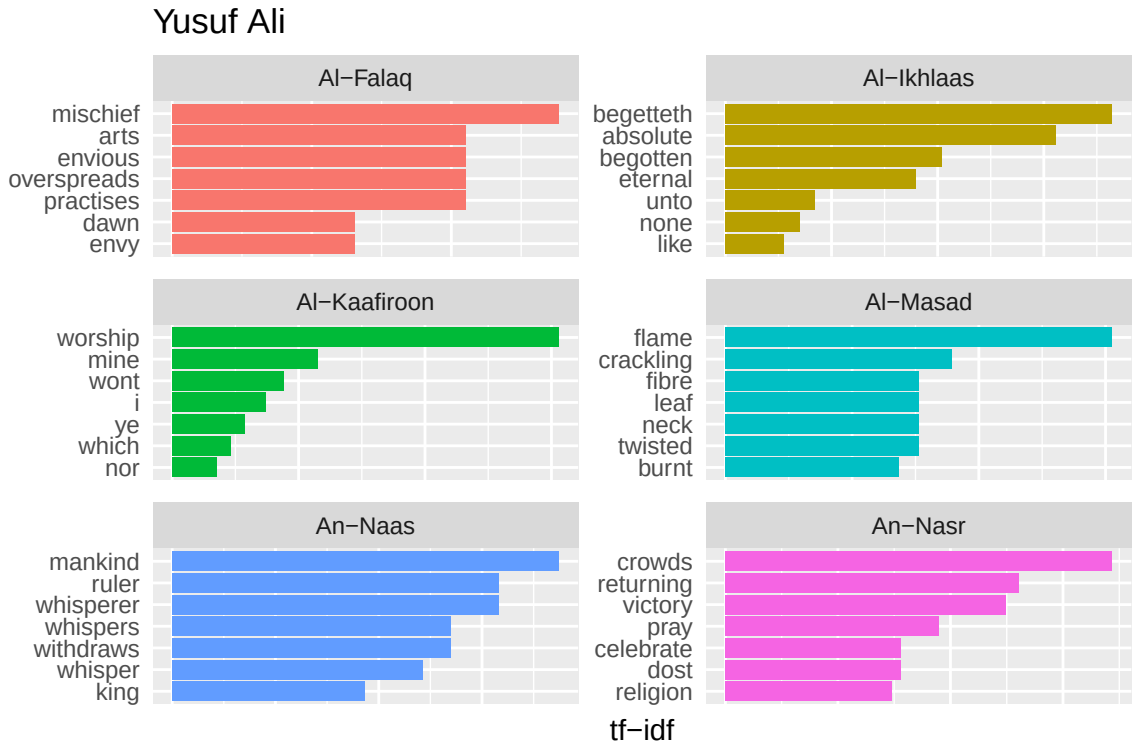


Figure 17: Short Surahs tf-idf for Yusuf Ali

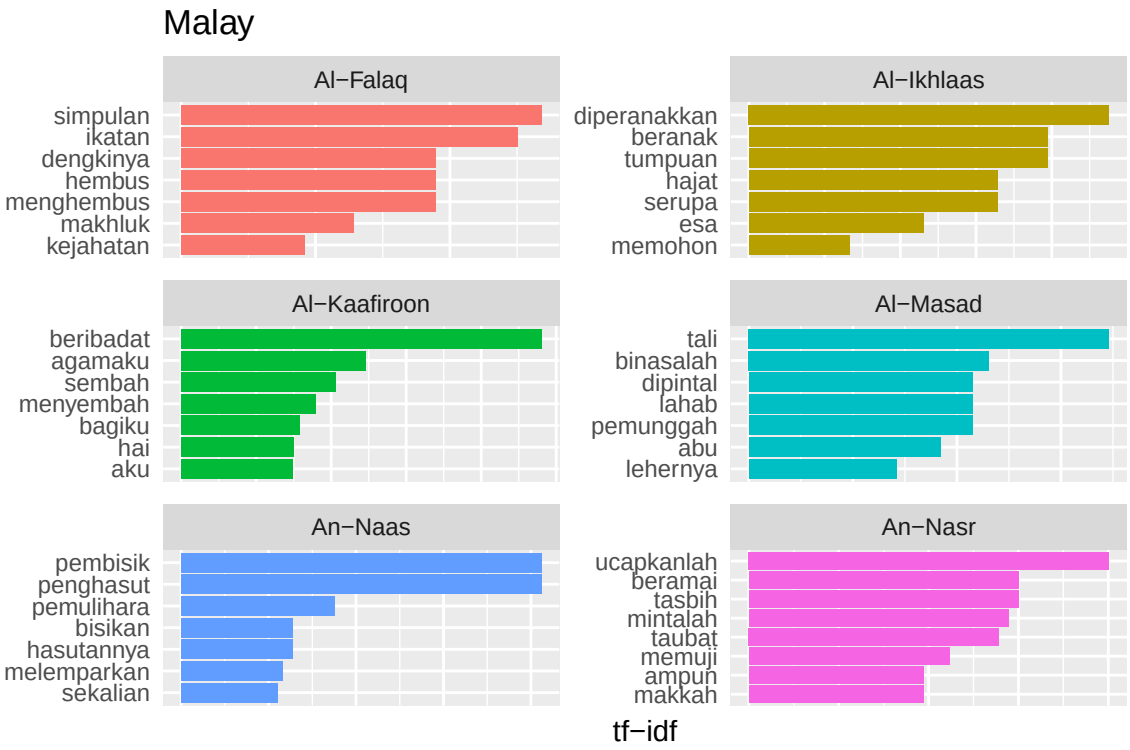


Figure 18: Short Surahs tf-idf for Malay

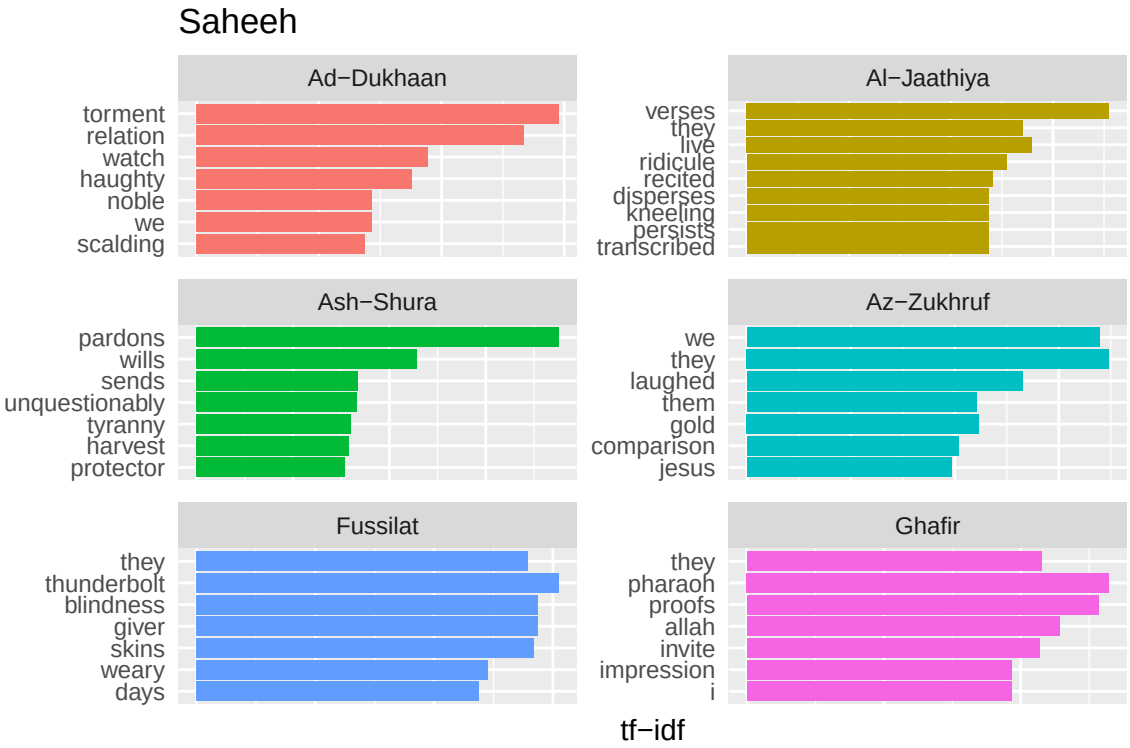


Figure 19: Medium Surahs tf-idf for Saheeh

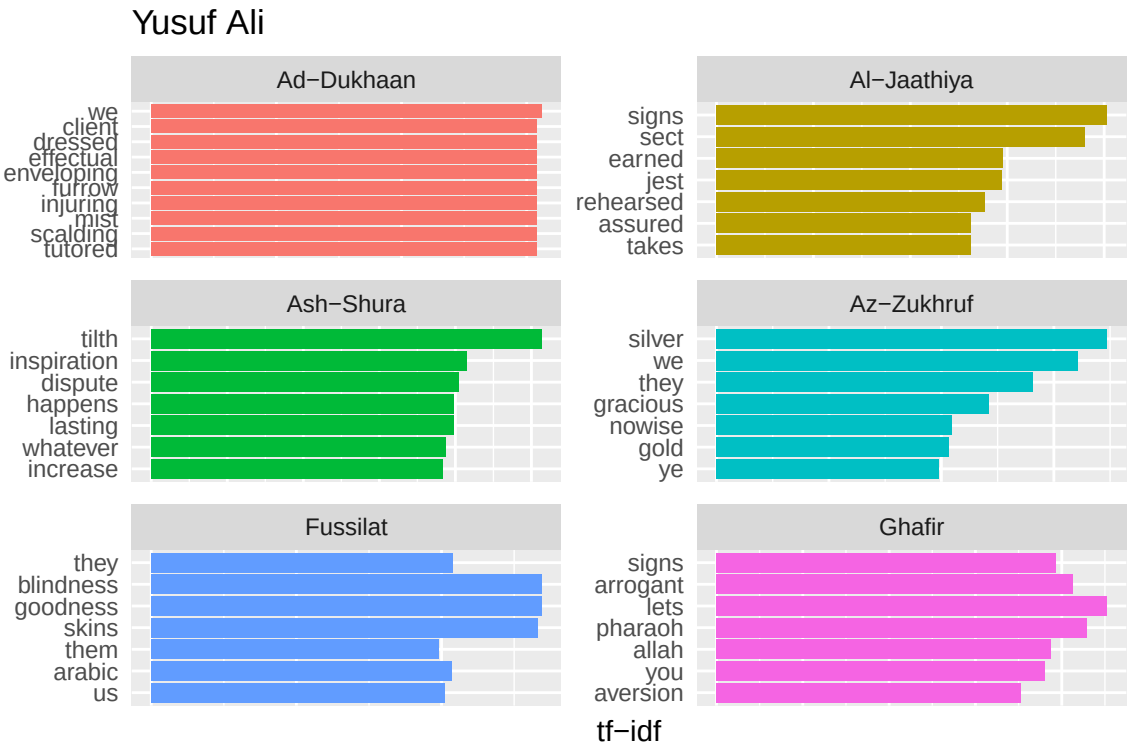


Figure 20: Medium Surahs tf-idf for Yusuf Ali

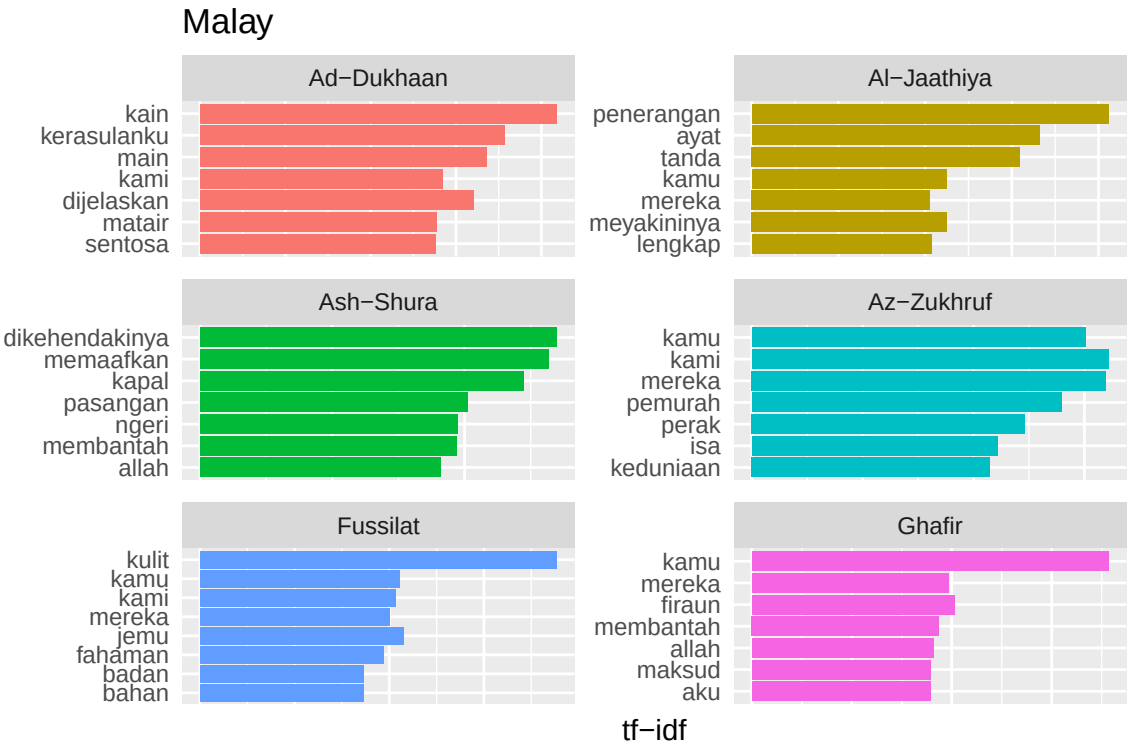


Figure 21: Medium Surahs tf-idf for Malay

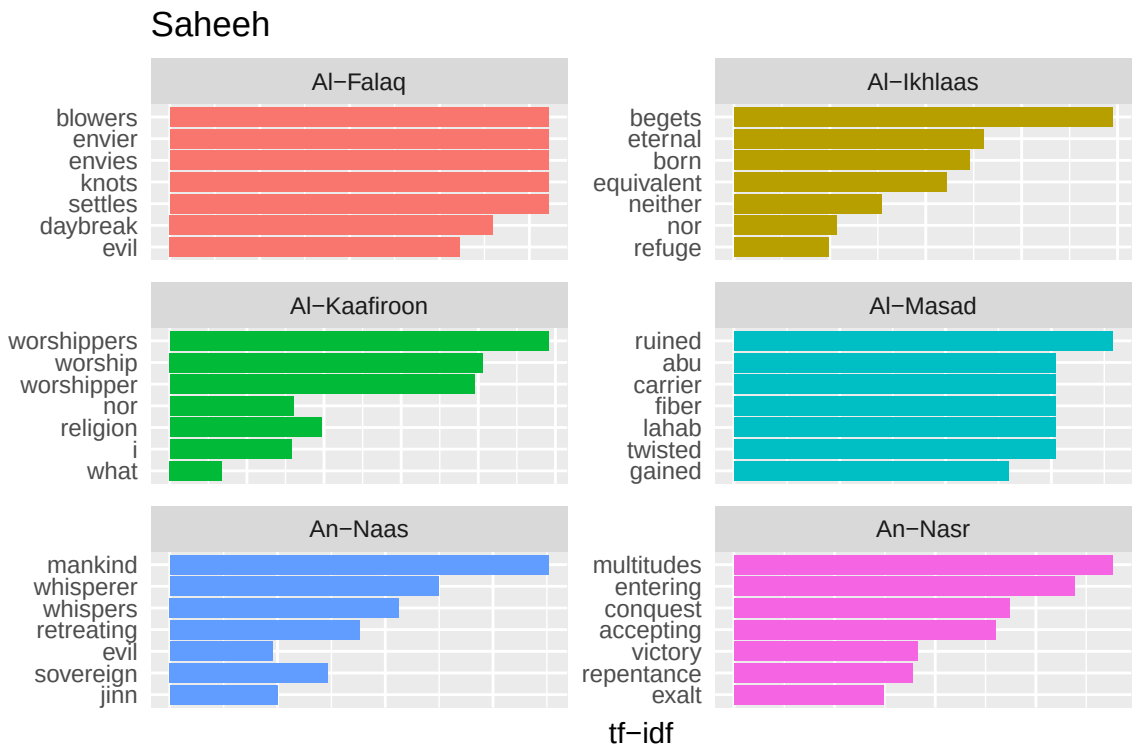


Figure 22: Long Surahs tf-idf for Saheeh

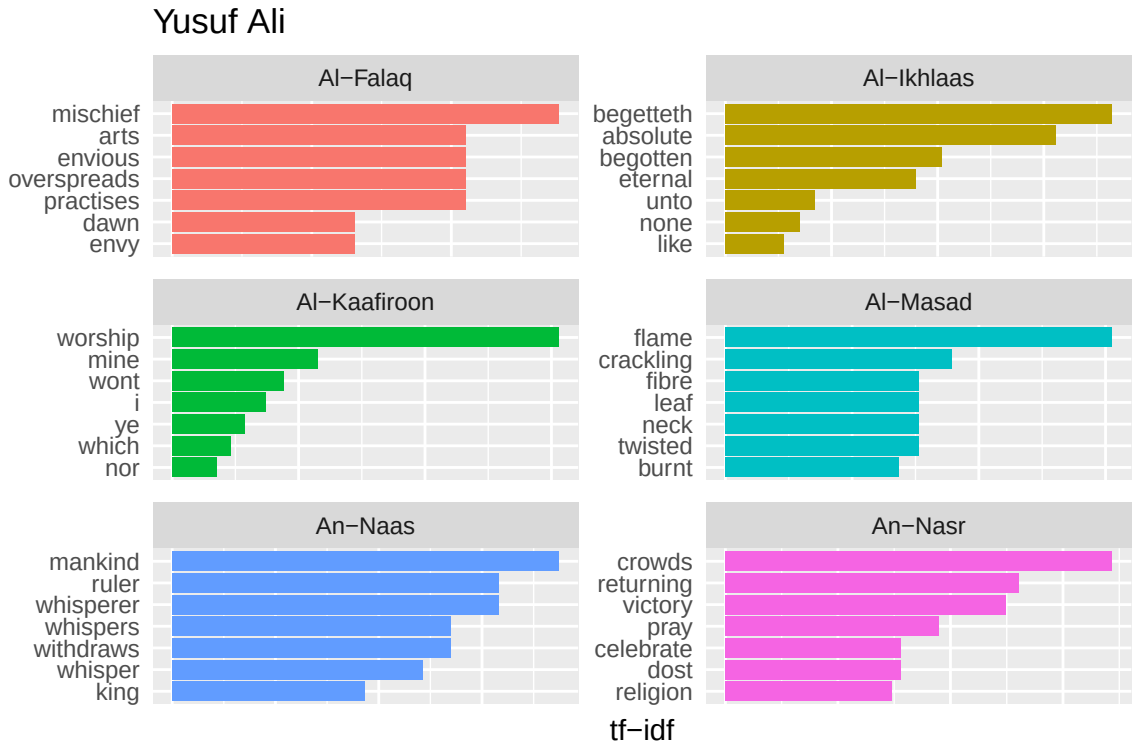


Figure 23: Long Surahs tf-idf for Yusuf Ali

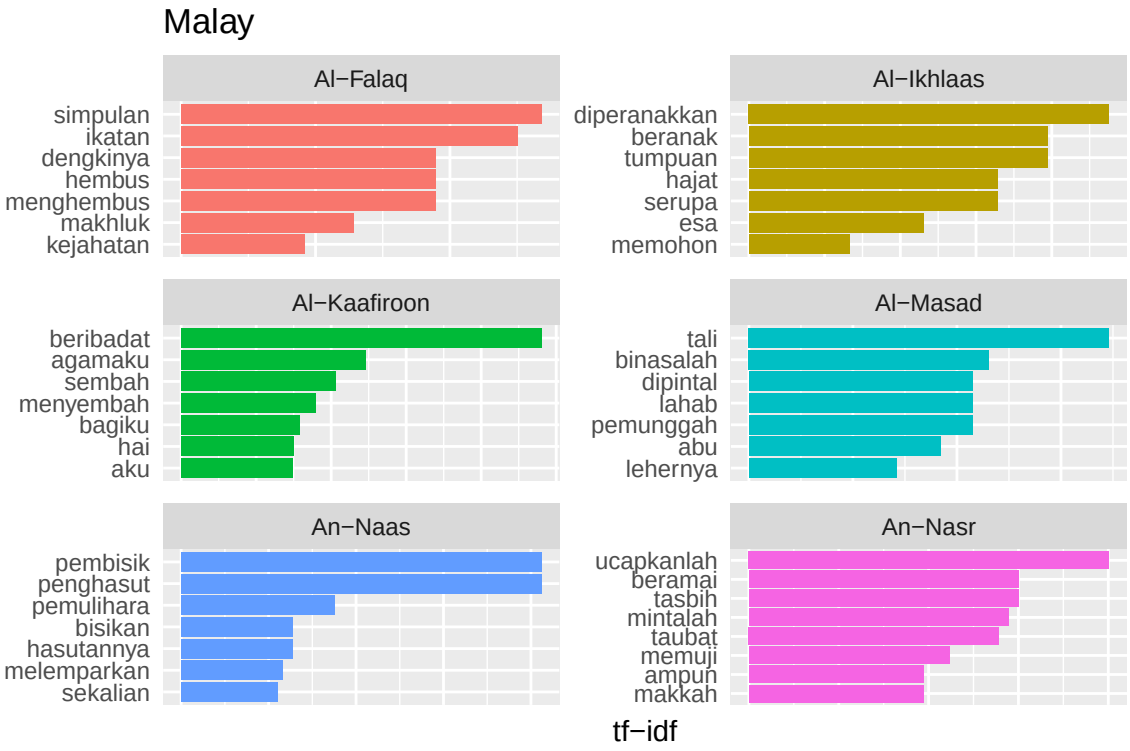


Figure 24: Long Surahs tf-idf for Malay

2.4 Zipf's law

Zipf's law states that the frequency of a word's appearance is inversely proportional to the rank of its frequency for a given corpus. In general, Zipf's law states that the frequency distributions of words in a corpus follow a Power Law behavior (Zipf, 1949). The words such as "allah" in the translations, for which the term frequency is high, are inversely related to its rank, which is low. A Power Law distribution from a statistical perspective is a distribution with both a fat tail and a long tail simultaneously. Power Law has many implications for statistical testing and scale-free network behaviors (a subject beyond the current discussion).

Here we present the Zipf's plots for the translations, indicating whether they follow Zipf's law (and hence power-law) and test the translations' similarities.

```
freq_by_rank_ESI <- surah_wordsESI %>%
  group_by(surah_title_en) %>%
  mutate(rank = row_number(),
         `term frequency` = n/total) %>%
  ungroup()
freq_by_rank_EYA <- surah_wordsEYA %>%
  group_by(surah_title_en) %>%
  mutate(rank = row_number(),
         `term frequency` = n/total) %>%
  ungroup()
freq_by_rank_MAB <- surah_wordsMAB %>%
  group_by(surah_title_en) %>%
  mutate(rank = row_number(),
         `term frequency` = n/total) %>%
  ungroup()
```

```
zipf_plotter = function(freq_rank_df,title_label){
  freq_rank_df %>%
  ggplot(aes(rank, `term frequency`, color = surah_title_en)) +
  geom_line(size = 0.5, alpha = 0.8, show.legend = FALSE) +
  geom_abline(intercept = -0.62, slope = -1.1,
              color = "black", linetype = 2) +
  scale_x_log10() +
  scale_y_log10() +
  labs(title = title_label,
       x = "log of rank",
       y = "log of term frequency")
}
p1 = zipf_plotter(freq_by_rank_ESI,"Saheeh")
p2 = zipf_plotter(freq_by_rank_EYA,"Yusuf Ali")
p3 = zipf_plotter(freq_by_rank_MAB,"Malay")
cowplot::plot_grid(p1,p2,p3, nrow = 1)
```

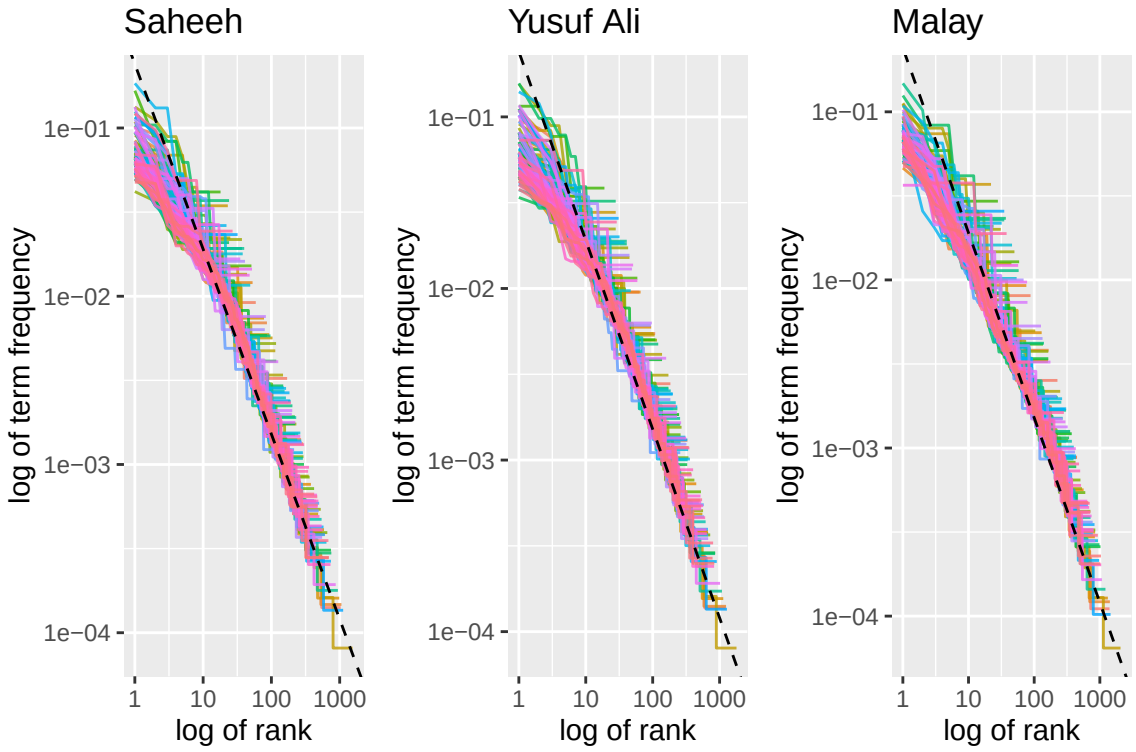


Figure 25: Zipf's plot

Note that the plot in Figure 25 is on a log-log scale (for testing Power Law), whereby if the lines are close to the inverse 45-degree line, then the law holds. We can see that all plots indicate a broad indication of adherence to Zipf's law. Based on this, we can conclude that all translations explain the same subject in broad terms, despite earlier indications of differences of styles. A visual test of the power-law is insufficient, and a full and proper test is required to prove this point, which is beyond the current scope of this book.

2.5 Words of high occurrence and stopwords

From the wordcloud plots in Figure 2, Figure 3, and Figure 4, we can see consistently the connecting terms emerged as high frequency terms for all versions of translation. The question is, how do we treat some words which occur in high frequencies, such as these connecting words, and at the same time, having low inverse ranking? Should these words remain in the texts or removed for the (statistical) analysis since their roles are more as word connectors rather than carrying any meaning? Is the decision to take out these words justified?

Most analyses of texts in English will take the option of removing these stopwords. While maybe acceptable in most circumstances, such an approach is not as straightforward when applied to Al-Quran translations. In the coming chapters, we will revisit the subject again. Let us redo Zipf's plot with all the stopwords removed and see what the removal of stopwords means.

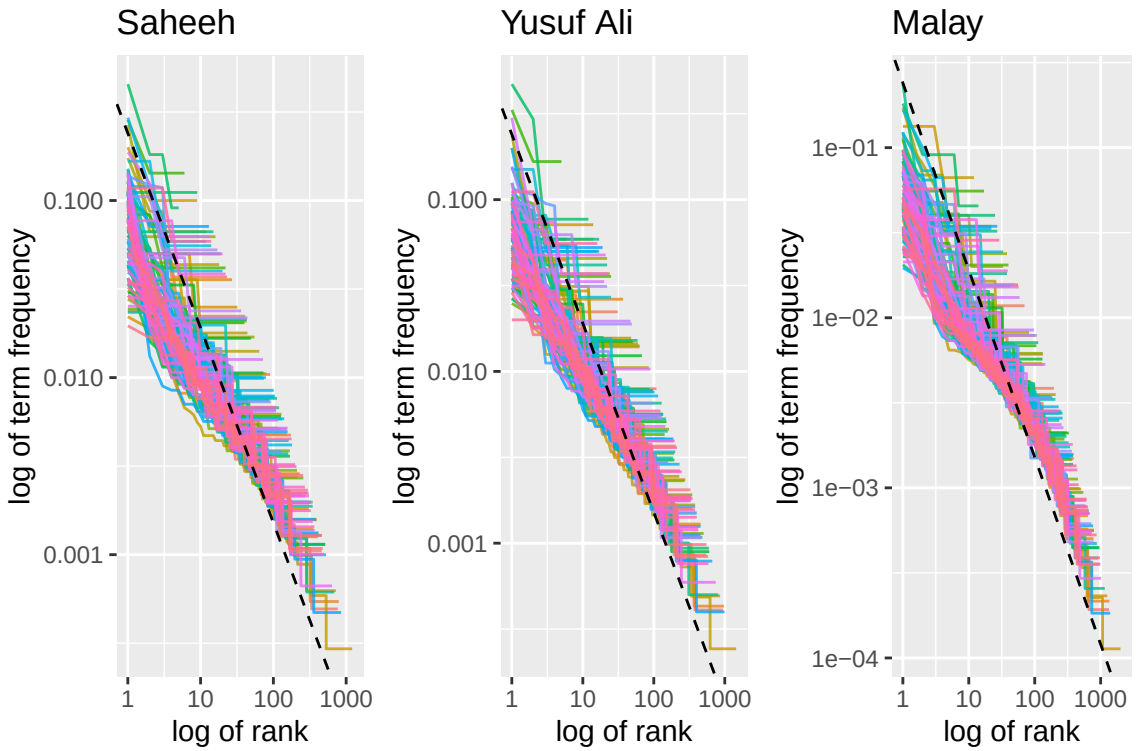


Figure 26: Zipf's plot without stopwords

From the plots in Figure 26, compared to the previous Zipf's plot in Figure 22, we can observe significant changes when we removed the stopwords. Putting the three translations together indicates that while removing the stopwords changes the texts' structure, it might not change the texts' meaning since the translations demonstrate similarities after their removal. In other words, the translations' texts are robust to changes due to these connector words. This is the nature of the English language as explained by Zipf (Zipf, 1949), and probably it is the same for the Malay language. However, are we sure that the meanings are retained? This must be proven beyond doubt for it to be conclusively accepted.

We have to caution the readers that we have been rather casual in some of the claims without rigorously providing statistical tests or proofs. Instead, we rely only on the visualizations provided and let the "data illustrate itself". Our purpose is to highlight various indications for future work and research and at the same time, allow readers with no statistical background, to understand the concepts and approach. The need for a full and proper test of the hypothesis as indicated above is one example case of precaution required before concluding.

2.6 Words of rare occurrence

Another method of analysis in word statistical analysis for corpus linguistics is to study **hapax legomenon** (singular) or **hapax legomena** (plural). It is about words that occur only once or extremely infrequently within the corpus. Examples of this in Al Quran are: "harut", "marut", and "zanjabil".[38] Studying hapax legomena helps us in understanding how different authors' (in our case here, translators) approach the usage of dictionary and vocabulary.

The subject of **hapax legomena** in linguistics requires a much more in-depth analysis, which we do not intend to perform in this book.[39] Here we will do a simple explanatory analysis demonstrating the uses of *hapax legomena* in text analysis.

The table below describes a summary of the number of (words) tokens and unique tokens (words). Please note the numbers for the Al-Quran Arabic for comparison.

Item	Al-Quran[40]	Saheeh	Yusuf Ali	Malay
Total number of tokens	77,430	158,065	167,859	204,784
Total number of unique tokens	18,994	5,251	6,365	7,156

Unique tokens are similar to vocabulary in some sense. It is the library of words used in the corpus. Comparing English and Malay versus Arabic, indicates clearly that the structure of language is starkly different. Arabic consists of much lower tokens (less verbose) and yet higher unique tokens (larger library), while Saheeh and Yusuf Ali are the opposite (more verbose, smaller library); and Malay on the other hand needs much more words (much more verbose) with smaller library than Arabic, and larger than the English corpora. (The Malay translation often includes explanations in the main text. That may explain the larger number of tokens used compared to the number of unique tokens.)

As we have described before, in the corpora, few words occur in high frequencies, and few words occur in very low frequencies at both extremes. We will plot the statistical density of the distribution (in log scale) of the term frequencies in both translations to observe this phenomenon.

The term frequencies (using a log scale for visualization) for both translations in Figure 27 show striking similarities, despite the style difference. On the left are the terms which occur infrequently, and as we add more terms to the vocabulary (as we move to the right), the density curve peaked at few distinct modes until it flattens out to the right as we bring into the vocabulary words frequently used. If we check what are the words on the left, examples would be "harut", "marut", "iram" (which are unique names), and in the middle are words like "food", "eat", "makan" (which are common words), and finally in the far right are words like "with", "the", "yang" (which are the stopwords).

An important observation is why the density curves have multiple "peaks"; there are two lower peaks on the left, and a few peaks in the middle, and a fat tail to the right. Is this structure common in any English corpus or Malay corpus? Or this is true only in the case of translations of Al-Quran. Comparisons between the corpora under study with other common corpora are required to confirm this phenomenon.[41]

The subject of vocabulary and rare word usage is essential and extensive by itself. In particular, term frequencies analysis of rare occurrence words may yield insights of its own, which revolves around why and how *hapax legomena* exists in communications and vocabulary development in languages. As a case in point, usage of words with multiple synonyms, such as "abrogate" versus "evade"; which one is a more appropriate choice for a given original Arabic word? Comparing these words against its original Arabic word would be

[38] https://en.wikipedia.org/wiki/Hapax_legomenon#Arabic_examples

[39] In R, there are a few packages which are useful for the analysis, a notable one is *qdap* (http://trinker.github.io/qdap/vignettes/qdap_vignette.html) which stands for "Quantitative Discourse Analysis Package".

[40] Dukes and Habash (2010)

[41] We will leave this subject as a research question.

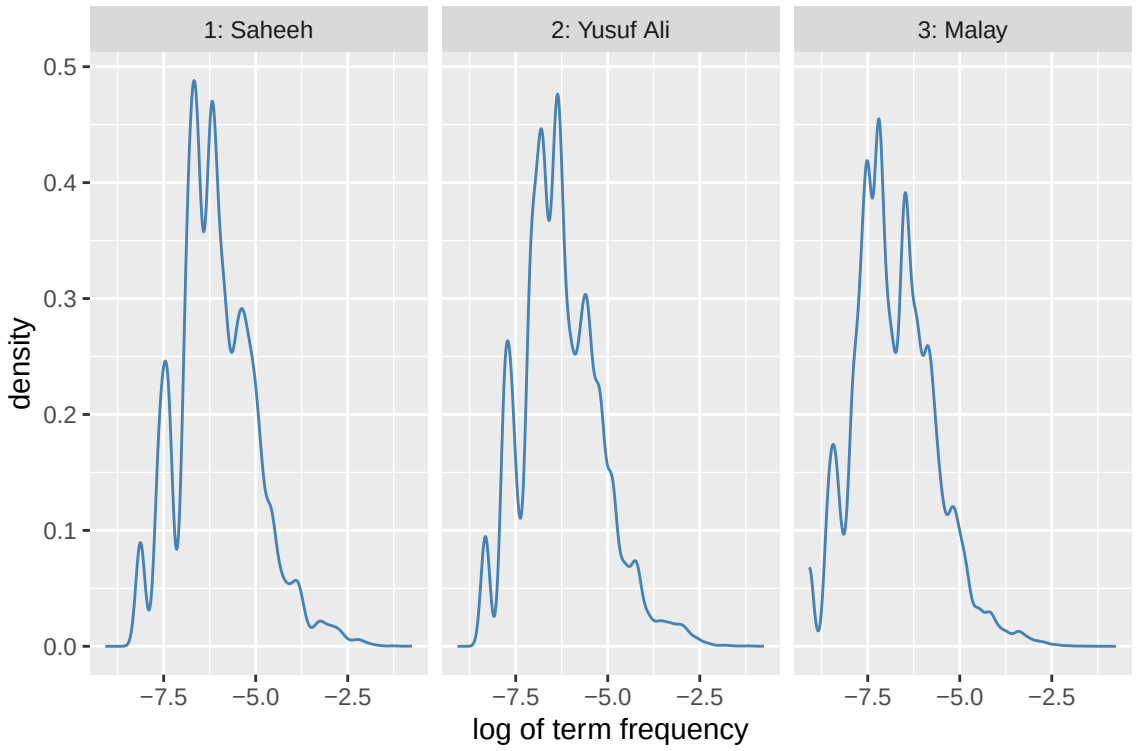


Figure 27: Density plot of term frequency

interesting since the Arabic may carry its distinctive meaning, while in contrast, the English word equivalent may carry its content or context.

We would like to show some of the types of analysis of rare words which may carry significant meanings, such as the word "trustworthy" and "trust". "Trustworthy" occurs only 13 times in Saheeh and 3 times in Yusuf Ali, while "trust" occurs only 3 times in Saheeh and many times in Yusuf Ali. The exact occurrences are shown below:

```
freq_by_rank_ESI %>% filter(word == "trustworthy")
freq_by_rank_EYA %>% filter(word == "trustworthy")
freq_by_rank_ESI %>% filter(word == "trust")
freq_by_rank_EYA %>% filter(word == "trust")
```

Only three occurrences of "trustworthy", in Surah Al-A'raaf, Al-Baqara, and Luqman, coincide in both translations. In Saheeh's Surah Ash-Shu'araa, it occurs six times and is not present in Yusuf Ali. Further checks reveal that in Yusuf Ali's Surah Ash-Shu'araa, instead of Saheeh's "trustworthy", "worthy of trust" is used in its place. Why Yusuf Ali prefers the word "trust" more than Saheeh is a subject worthy of understanding by itself. Furthermore, as we can see from the outputs, the tf-idf measures reveal the words' positions within the Surah and the corpus, which is an indicator of its own understanding.

We showed here many dimensions of analysis using words, word frequencies, inverse frequencies, word positions within a sub-set of texts (i.e., Surahs), and the whole corpus, across corpora. These analyses are rich in understanding and knowledge of linguistics, meanings, and contexts. We also showed the intricacies involved in word selection for translations like the case for "trust", "trustworthy", and "worthy of trust".

2.7 Words with medium occurrence

What do words of non-high occurrence and non-rare occurrence, hence medium occurrence, represent? For once we know that these words represent the largest part of the vocabulary library for each corpus.

Let us take Saheeh as our example. The highest rank is "allah" (rank = 1), and among the lowest rank is "wombs" (rank = 1199). Let us check what are the words in the middle rank, rank around 600.

```
freq_by_rank_ESI %>% filter(rank > 601 & rank < 608 ) %>% head()
```

We can see words, such as "oaths", which describe an action, occurs 13 times in the Saheeh corpus. Similarly the name of the Prophet Noah, which occurs 28 times; which case is different since the word is a special name. Another word, such as "masjid" which generally means the mosque, occurs 9 times. We can see that these middle-frequency words are non-trivial since they do hold a special purpose within a certain context and meaning; for example, we have a very strong action, which is an oath, a special name, which is the Prophet Noah, and a special place, a Mosque.

Words of mid-frequency occurrence become very important when we need to go deeper into the context and meaning of words, sentences, groups of sentences (such as Surahs) - a subject that we will revisit in later chapters.

2.8 Summary

While seemingly simple, statistical word analysis yields many insights into issues relating to a corpus of texts. In this chapter, we have shown examples of how the analysis is useful for many linguistic studies of Al-Quran translations. We also compared different translations of the English language (Saheeh and Yusuf Ali) and different languages (English and Malay).

Analysis using term frequency (tf) and inverse document frequency (idf) allows us to find word characteristics for one document within a collection of documents. Exploring term frequency on its own can give us insight into how language is used in a collection of natural language and give us tools to reason about term frequency. The proper noun "Allah" ranks very high on almost all the statistics of the English Quran which confirms that "Allah" is the central and most important subject matter of the Quran, a topic that one of the authors will cover in an upcoming book (Alsuwaidan and Hussin, 2021).

Statistical analysis of words from the Quran is a good and "easy" start to Quran Analytics. It is general and robust, requires no or little manual effort, and is "surprisingly" powerful. As we have indicated in this chapter's various suggestions, the subject requires further research. Among the issues raised is about translations of Al Quran into another language, such as the English and Malay language (as the case here), which has its structural dependencies. Will the usage of different structure results in a difference in meaning and understanding? Is there any better way to detect and compare different writing styles to ensure or reflect accurate meaning to the readers? How does the usage of infrequent words add to the vocabulary richness of the texts? The words statistical analysis is a starting point to answer these types of questions and many other questions a researcher of Al-Quran may explore.

We leave this chapter and many of the issues raised for future research works. The following is an incomprehensive list of what's possible:

1. Statistical NLP, which is based on word frequencies, has a lot to offer. Within a language, across languages, for purposes of translations between languages, are open examples. Al-Quran, a sacred text for the Muslims, must be translated with care. The various authors have done all existing translations based on their styles and understanding of both the original texts (in Arabic) and the language of translation. There is a need to revisit this subject using NLP tools to provide a reflective meaning suitable for the current time.

2. Word frequency analysis is among the simplest tool available yet provides many insightful understandings of language. We also covered Zipf's law, scale-free, power-law distributions of word frequencies. We showed how words of high occurrence, which are stopwords and words of importance, reveal many insights about the language and the messages in texts; while words of extremely low occurrence, *hapax legomena*, direct towards vocabulary usages and special meanings. All of these mentioned items can be extended as a research subject by themselves.

3. An example of classical work in Arabic for Al-Quran word-by-word references is *Al-Mu'jam Al-Mufahras Li Alfaz Al-Quran Al-Kareem* by Muhammad Fu'ad Abdul-Baqi (Abdul-Baqi (1945)). If the works in *Al-Mu'jam* are converted to tokens' data structures as we have done for the English or non-Arabic texts, cross-referencing can be quickly done. Using this tool, any Quranic scholar can check the accuracy or appropriateness of the Quranic translations with ease. It is among the examples of why we need the tools of NLP for Quran Analytics.

4. The simplest tools of data science, namely simple statistical tests, have not been explored to their full potentials. The power of data visualizations using programming languages such as **R**, as we have done here, is enormous.

The room to improve and expand on "word frequency analysis" is tremendous. We are just at its beginning stage.

2.9 Further readings

Abdul-Baqi, M. F. *Al-Mu'jam Al-Mufahras Li Alfaz Al-Quran Al-Kareem.* Dar Ahi'a Al-Tirath Al-Arabi, Beirut, Lebanon, 1945. (Abdul-Baqi, 1945)

Alsuwaidan, T. and Hussin, A. *Islam Simplified: A Holistic View of the Quran.* To be published manuscript, 2021 (Alsuwaidan and Hussin, 2021)

Zipf, G. K. *Human Behavior and the Principle of Least Effort.* Addison-Wesley, New York, New York, 1949. (Zipf, 1949)

tidytext package in **R**. (Queiroz et al., 2020)

ggplot2 package in **R**. (Wickham et al., 2020)

quRan package in **R**. (Heiss, 2018)

Tidy text mining website: https://www.tidytextmining.com/tidytext.html

3 Word Scoring Analysis

> "Indeed, We have sent you, [O Muhammad], with the truth as a bringer of good tidings and a warner, and you will not be asked about the companions of Hellfire." [Al-Quran Saheeh 2:119]

> "And We have not sent you except as a giver of glad tidings and a warner to all Mankind, but most of mankind know not." [Al-Quran Saheeh 34:28]

Once we have organized the texts into words (i.e., tokenize), we are ready to perform further analysis dealing with the words as data, in scoring these words following a scoring model. The selected scoring model will depend on the objective of the analysis and the structure of the model used. One such method is called *sentiment analysis*.

In this chapter, we will explore sentiment analysis on the Saheeh and Yusuf Ali English translations of Al-Quran. We will not do any analysis on the Malay version because we could not find any reliable sentiment model developed for the Malay language. Sentiment analysis is done using existing *sentiment scoring models* or *sentiment lexicons* for the English language. The choice of which scoring models to choose depends on the objective of the analysis. Since we are using these pre-built models, we do not assume any accuracy of the analysis. However, as an explanatory exercise, we will use them to demonstrate the comparison between the two English translations of Al-Quran (Saheeh and Yusuf Ali) and a general indication of usages of scoring models in NLP, of which sentiment analysis is a special case.

One fundamental theme of Al-Quran is "Basheeran" and "Nazeeran", bringing "glad tidings" and "warnings" to all mankind, as denoted by verse 2:119 and verse 34:28, quoted above.[42] From a sentiment analysis perspective, "glad tidings" is equated with "positive sentiments" and "warnings" is equated with "negative sentiments". In this chapter, we will explore sentiment analysis to view how these sentiments are reflected in the English translations of Al-Quran.

We will explore a few types of sentiment scoring models and few methods of application to enable us to understand how sentiment analysis work. Any model will have its own assumptions, hence results may differ based on using different models. One of the reasons why this is the case depends on how the scoring model was derived. For example, the *bing* model is based on "-1" and "+1" scores; the *nrc* model is based on a binary of "yes" and "no"; while the *AFINN* model is based on scores between -5 to +5. In reality, we can build our own sentiment scoring model given enough datasets (or collection of corpus), based on the subject concerned. This subject is beyond the scope of the current work and left for future research.

Since we will use the same dataset that as in Chapter 2, and the *tidytext* package, readers can assume that the computing environment in Chapter 2 is continued here.

3.1 Preprocessing the data

All the works in **R** for this chapter are just a continuation of the previous computing environment. Before we proceed, we have to decide whether to remove stopwords from the texts or to retain them. The reasons for removing stopwords are explained in Chapter 2, to remove frequently occurring words that carry no further meaning, except for continuation of the sentence. In *tidytext* the English stopwords are pre-prepared from three lexicons as *stop_words* variable. We can use them all together, as we have here, or use the *filter()* functions to only use one set of stop words if that is more appropriate for certain analysis.

```
stop_words = tidytext::stop_words
tidyESIC <- tidyESI %>%
  anti_join(stop_words)
ya_stop_words <- rbind(stop_words,
                       c('ye', 'verily', 'will', 'said',
                         'say', 'us', 'thy', 'thee',
                         'thou', 'hath', 'doth'))
tidyEYAC <- tidyEYA %>%
  anti_join(ya_stop_words)
```

Now let us compare the words (tokens) before and after the removal of stopwords:

Translations	Total tokens before	Total tokens after	Unique tokens before	Unique tokens after
Saheeh	158,065	43,996	5,251	4,783
Yusuf Ali	167,859	52,409	6,365	5,862

We can see that the words consist of 72.17 percent of stopwords in Saheeh, and 68.78 percent in Yusuf Ali. That is a significant number of words. On the other hand the stopwords tokens removed are only 8.91 percent for Saheeh, and 7.9 percent for Yusuf Ali, which is much lesser in percentage terms.

Since the data has been cleaned from stopwords, we are now ready to analyze them by having a quick look at the top common words which occur, say, more than 150 times. This is shown below for both translations.

```
ch3_plotter1 = function(df,title_label) {
      df %>%
      count(word, sort = TRUE) %>%
      filter(n > 150) %>%
```

[42] Verses with "Basheeran" (glad tidings) appears 18 times, and "Nazeeran" (warnings) appears 58 times. By the number of counts of words, the emphasis of "warning" is greater than "glad tidings".

```
      mutate(word = reorder(word, n)) %>%
      ggplot(aes(x = word, y = n)) +
      geom_col() +
      labs(title = title_label, x = NULL, y = "number") +
      coord_flip() +
      theme(axis.text = element_text(
            angle = 0,
            color="blue",
            size=10))
}
```

```
p1 = ch3_plotter1(tidyESIC,"Saheeh")
p2 = ch3_plotter1(tidyEYAC,"Yusuf Ali")
cowplot::plot_grid(p1,p2)
```

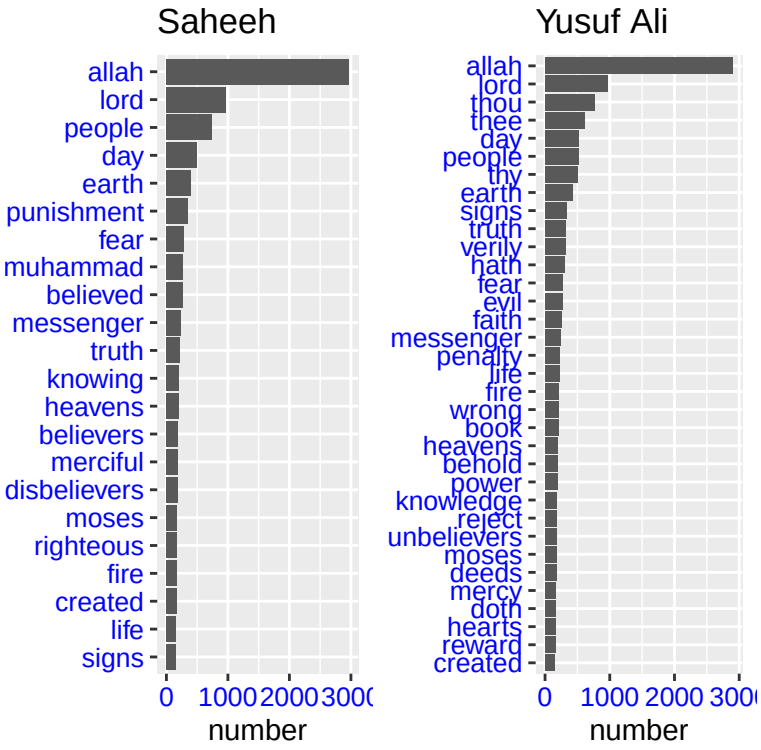


Figure 28: Top common words in the translations

Now we can see from Figure 28 that the term "Allah" is indeed the highest occurring term by a large count, followed by the term "Lord". This is not surprising since both terms are indeed part of one major theme of Al-Quran. An interesting observation is the term "Muhammad" (SAW) does not appear in Yusuf Ali among the top terms but appears in Sahih. However, the term "messenger" appears in both with different rankings. There are other observations on other terms or words; since this is not the focus of the current analysis, we will leave it to the readers to make their own observations.

3.2 Sentiment analysis with tidy data

In Chapter 2, we explored the tidy text format and showed it can easily be used to approach questions about word frequency in the English Quran. This allowed us to analyze which words are used most frequently in the Quran and to compare two versions of the English Quran.

Now let us address the topic of opinion mining or sentiment analysis. We can use the tools of text mining to approach the emotional content of text programmatically.

One way to analyze the sentiment of a text is to consider the text as a combination of its individual words and the sentiment content of the whole text as the sum of the sentiment content of the individual words. There are other approaches but this approach can easily take advantage of the tidy tools.

3.2.1 Sentiment scoring models

The *tidytext* package provides access to several sentiment lexicons. Three general-purpose lexicons are

1. AFINN from Finn Årup Nielsen,
2. bing from Bing Liu and collaborators, and
3. nrc from Saif Mohammad and Peter Turney.

All three of these lexicons are based on unigrams, i.e., single words. These lexicons contain many English words and the words are assigned scores for positive/negative sentiment, and also possibly emotions like joy, anger, sadness, and so forth. In this section, we will use the *bing lexicon.*

The function *get_sentiments()* allows us to get specific sentiment lexicons. An example is as follows:

```
get_sentiments("bing")
```

3.2.2 *bing* scoring model

One advantage of having the data.frame with both sentiment and word is that we can analyze word counts that contribute to each sentiment. By implementing *count()* here with arguments of both word and sentiment, we find out how much each word contributed to each sentiment.

This can be shown visually, and we can pipe straight into *ggplot2*, if we like, because of the way we are consistently using tools built for handling tidy data frames. Now let us plot the main words which have negative and positive sentiments for both translations.

```
bing_word_counts_ESIC <- tidyESIC %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  ungroup()
bing_word_counts_EYAC <- tidyEYAC %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  ungroup()

ch3_bing_plot = function(sent_df,title_label){
  sent_df %>%
    group_by(sentiment) %>%
    top_n(20) %>% ungroup() %>%
    mutate(word = reorder(word,n)) %>%
    ggplot(aes(word,n,fill = sentiment)) +
      geom_col(show.legend = FALSE) +
      facet_wrap(~sentiment, scales = "free_y") +
      labs(y = title_label, x = NULL) +
      coord_flip() +
      theme(axis.text = element_text(
```

```
                          angle = 0,
                          color = "blue",
                          size = 10))
}
p1 = ch3_bing_plot(bing_word_counts_ESIC,"Saheeh")
p2 = ch3_bing_plot(bing_word_counts_EYAC,"Yusuf Ali")

cowplot::plot_grid(p1,p2)
```

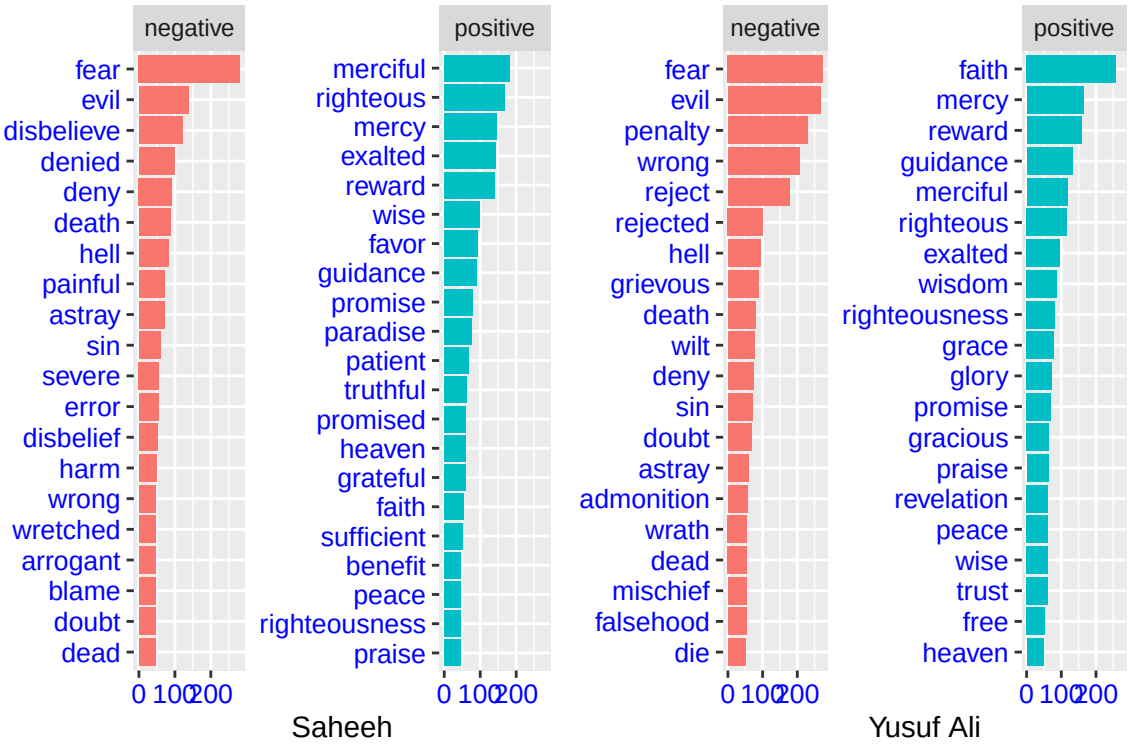


Figure 29: Bing's top negative and positive words

Figure 29 interestingly shows that for both translations, the shape is identical for both negative and positive plots. Again, we can see that the top two words for negative are the same for both translations, "fear" and "evil", but for positive they differ: "merciful" and "righteous" for Saheeh, and "faith" and "mercy" for Yusuf Ali. The scoring for other words differs quite significantly, which implies that the style of writing for the translations is not the same.

This indicates that while translating from the same source, using the *bing* sentiment model, the sentiments expressed by the words in English as translated, reflect different intonations from a sentiment point of view. Whether this is due to the sentiment model or the writing is not known, until further investigations. The way to do this is by running the same analysis using a different scoring model. The results show that the scoring is "model-dependent", as we will show next.

3.2.3 *AFINN* scoring model

Now let us compare with the *AFINN* scoring model and observe if there are any major changes to the findings using the *bing* model.

We can see that the dimensions of sentiment are different in the *AFINN* model as compared to *bing*. Furthermore, the scoring numbers and the terms, as well as the counts for the two translations, show

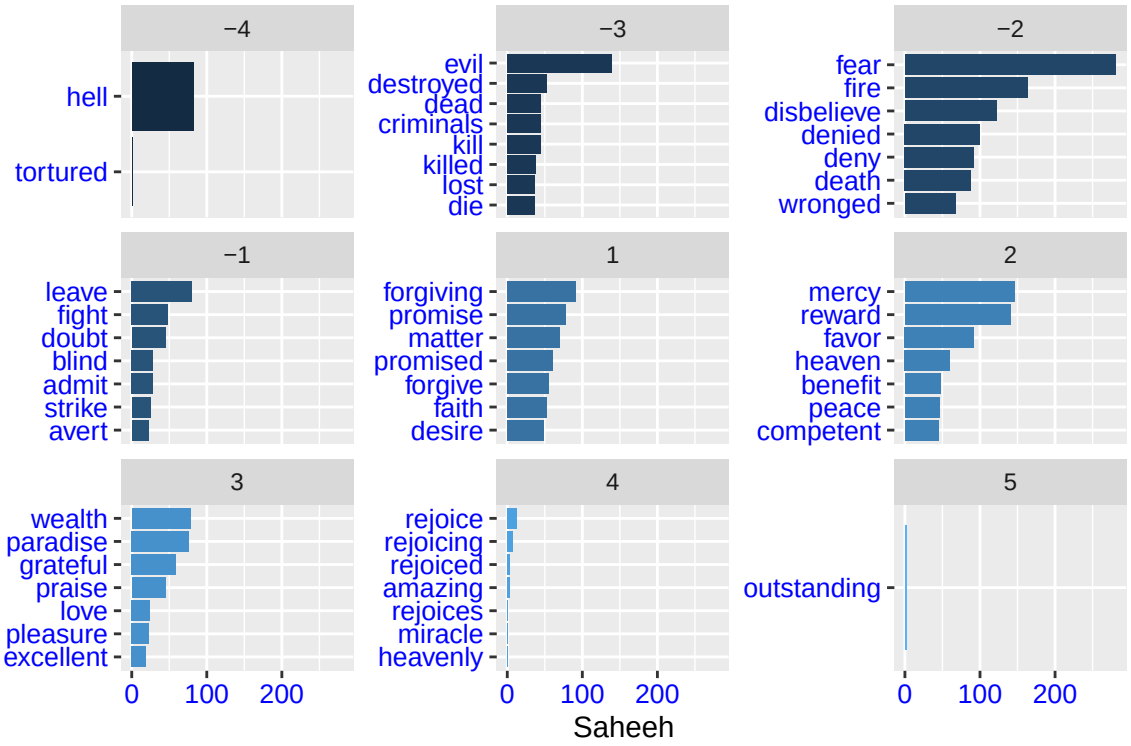


Figure 30: AFINN scoring model on Saheeh

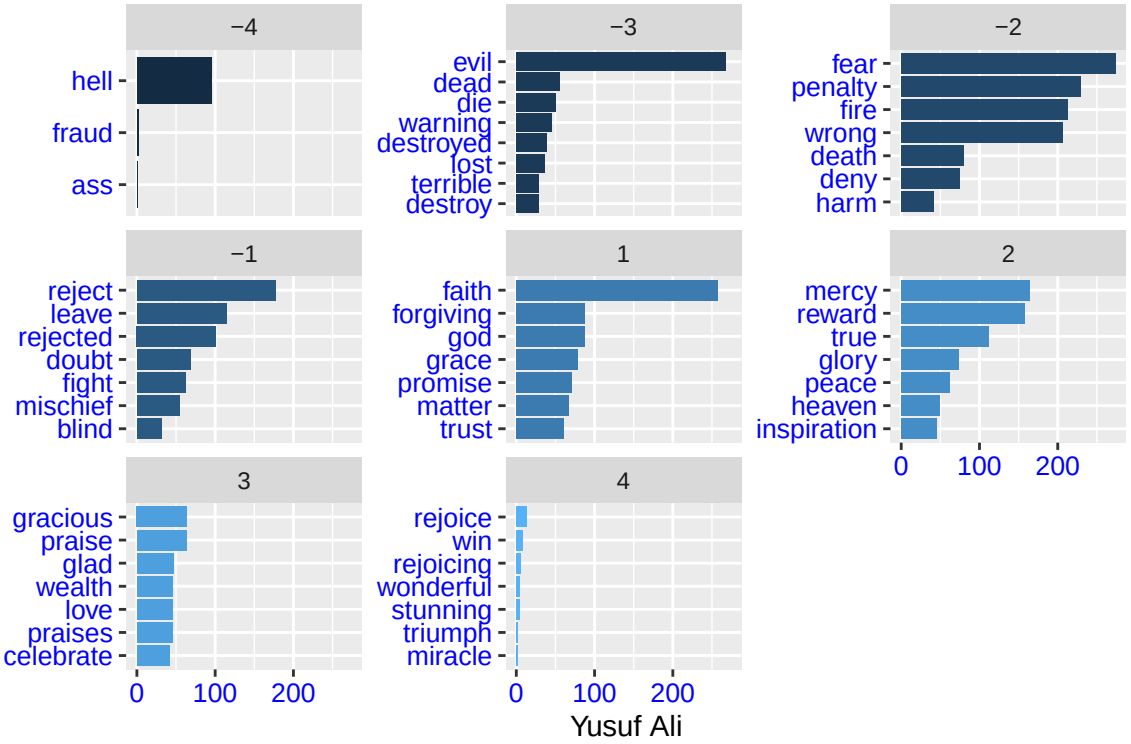


Figure 31: AFINN scoring model on Yusuf Ali

major differences (as observed from Figure 30, the plots for Saheeh compared to Figure 31, the plots Yusuf Ali). Again we want to note that what we have shown is just an exploratory view of the sentiment analysis using various models and what they may imply.

There are many other ready-made sentiment scoring models available. We will leave it to the readers to try on their own.

3.3 Sentiment analysis within the Surahs

Since Al-Quran is arranged by Surahs or chapters, we would like to investigate if we can use sentiment scoring models to score each of the Surahs and measure which Surahs use the most "negative" words ("warnings") and the most "positive" words ("glad tidings").

We can apply the *bing* model as follows:

```
bingnegative <- get_sentiments("bing") %>%
  filter(sentiment == "negative")

wordcountsESIC <- tidyESIC %>%
  group_by(surah_title_en) %>%
  summarize(words = n())

wordcountsEYAC <- tidyEYAC %>%
  group_by(surah_title_en) %>%
  summarize(words = n())

ch3_bing_plot2 = function(sent_df,title_label){
  sent_df %>%
      semi_join(bingnegative) %>%
      group_by(surah_title_en) %>%
      summarize(negativewords = n()) %>%
      left_join(wordcountsESIC, by = c("surah_title_en")) %>%
      mutate(ratio = negativewords/words) %>%
      top_n(20) %>%
      ggplot(aes(x = surah_title_en, y = ratio)) +
        geom_col() +
        labs(title = title_label, x = NULL) +
        coord_flip() +
        theme(axis.text = element_text(
              angle = 0,
              color="blue",
              size=10))
}

p1 = ch3_bing_plot2(tidyESIC,"Saheeh")
p2 = ch3_bing_plot2(tidyEYAC,"Yusuf Ali")

cowplot::plot_grid(p1,p2)
```

Figure 32 shows the Surahs or chapters with the most "negative" words, normalized for the number of words in the Surah. The Surahs in Saheeh differ from the ones in Yusuf Ali. There are some similarities and differences between the translations, which very likely is dependent on the exact words used in the Surahs and the scores applied.

Let us repeat for "positive" Surahs.

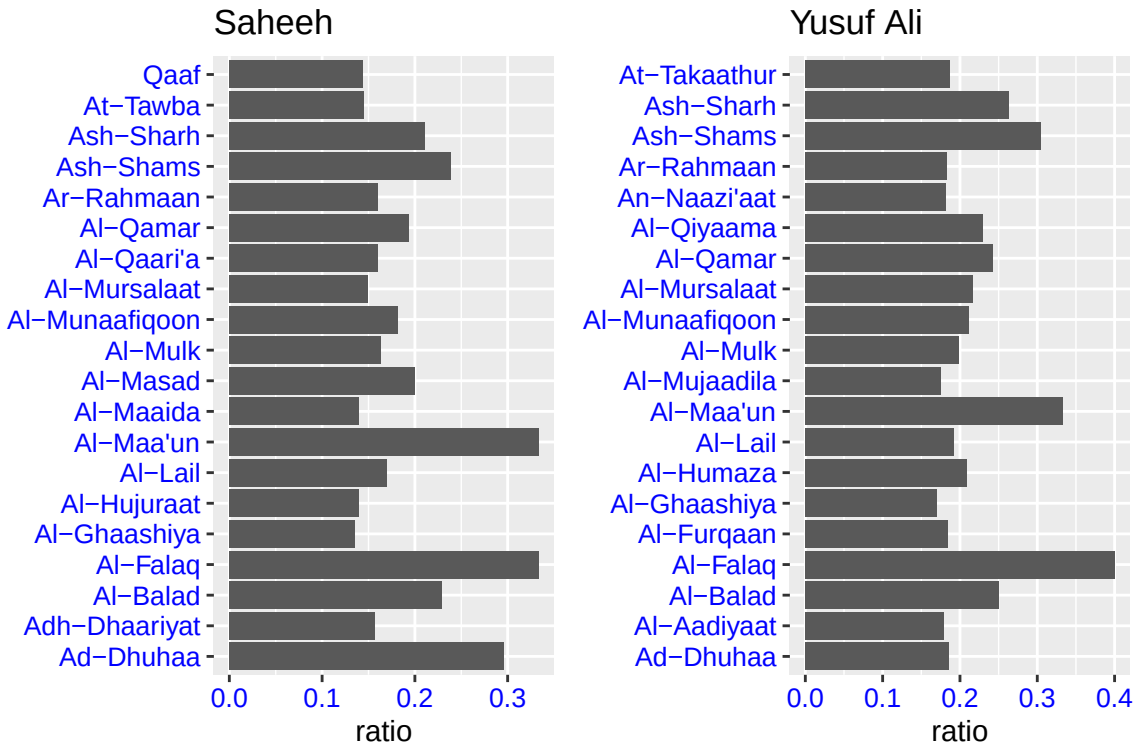


Figure 32: Negative Surahs scoring

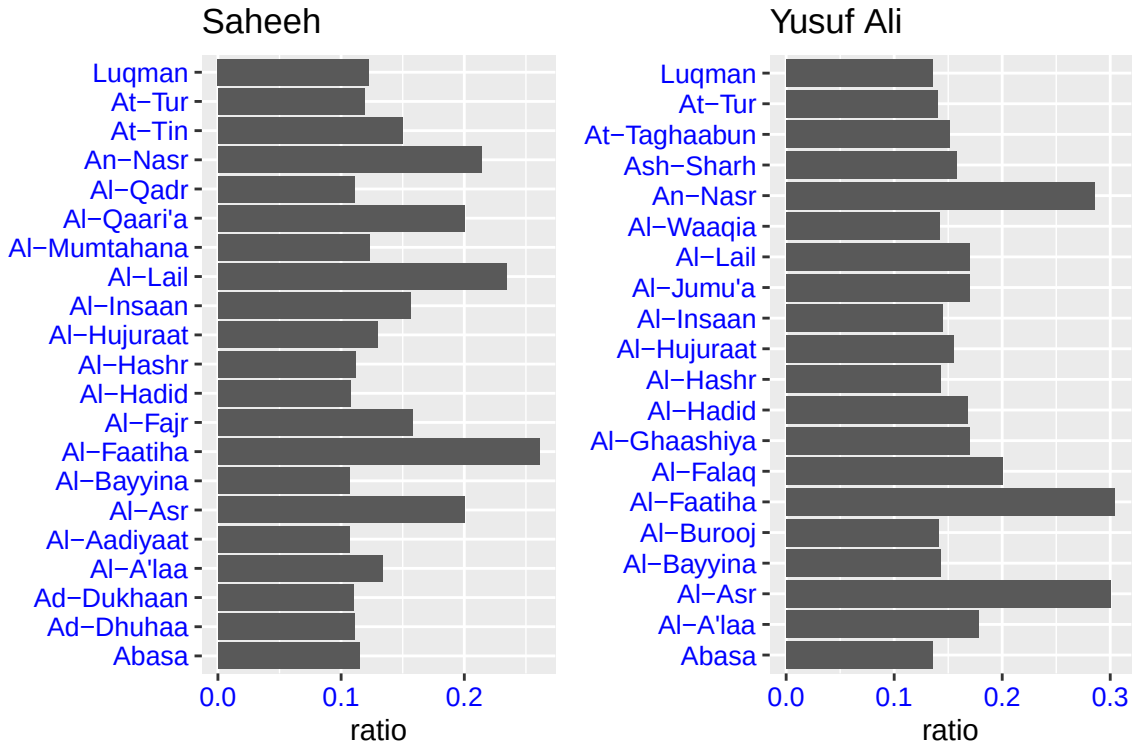


Figure 33: Positive Surahs scoring

It is interesting to note the similarities and differences between Saheeh and Yusuf Ali for the categorization of Surahs as demonstrated in Figure 33.

Now let us study the sentiment within a Surah, such as Surah Yusuf, which is a middle-length Surah for comparison. The idea here is to observe the sentiment scores as we move along the Surah.

```
Surah_yusuf_ESIC <- tidyESIC %>%
  inner_join(get_sentiments("afinn")) %>%
  filter(surah_title_en %in% c("Yusuf"))

Surah_yusuf_EYAC <- tidyEYAC %>%
  inner_join(get_sentiments("afinn")) %>%
  filter(surah_title_en %in% c("Yusuf"))
ch3_syplot = function(sy_df,title_label,col_disp){
  sy_df %>%
        ggplot(aes(x = 1:nrow(sy_df), y = value )) +
          geom_col(show.legend = FALSE, color = col_disp) +
          labs(title = title_label,
               x = "word order",
               y = "sentiment score")
}

p1 = ch3_syplot(Surah_yusuf_ESIC,"Saheeh","magenta")
p2 = ch3_syplot(Surah_yusuf_EYAC,"Yusuf Ali","darkgreen")

cowplot::plot_grid(p1,p2)
```

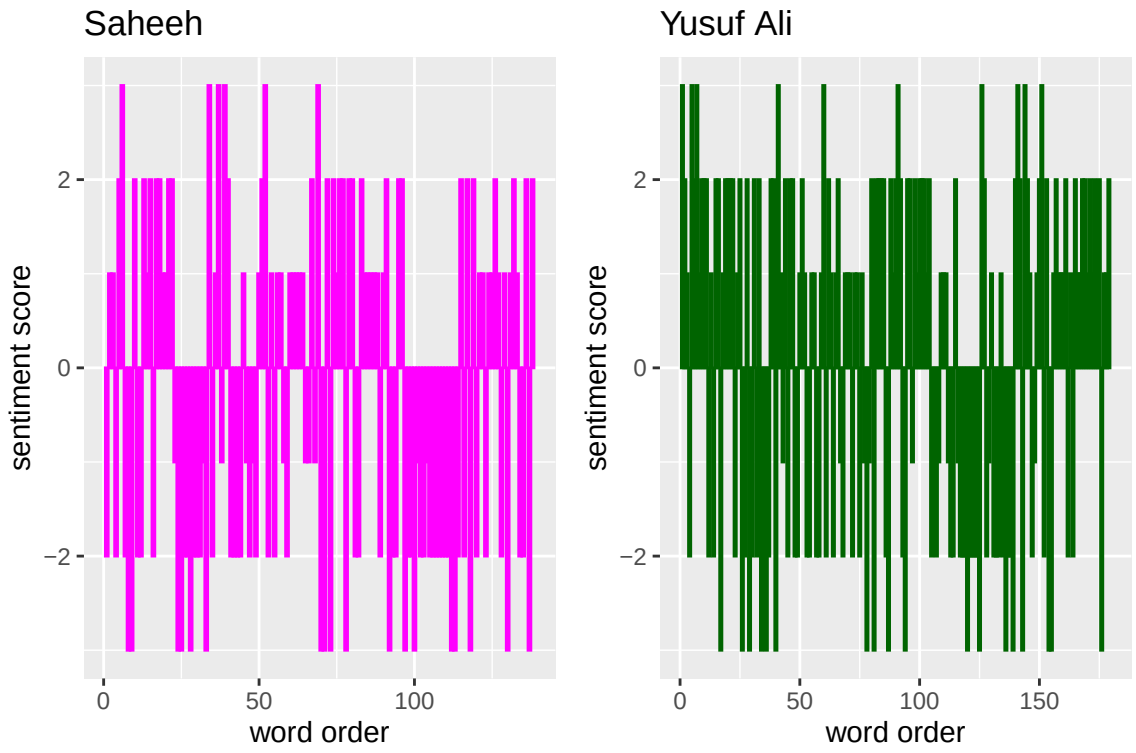


Figure 34: Sentiment scoring within Surah Yusuf

Here in Figure 34, we use the *AFFIN* model for scoring and tracking the scores as we move from one word

to the next in Surah Yusuf. As we can see from the plot, the sentiments (as indicated by the scores), change alternatingly like a wave throughout the Surah, from a series of positives to a series of negatives. This is the narrative view of the sentiments within the flow of the texts. This observation deserves some attention since, in most normal texts such as novels, the sentiments are not as alternating as we see here.[43] If we take the meaning of giving "good tidings" (Basheeran) and "warnings" (Nazeeran) of Al Quran, the alternating scores probably reflect it. The same analysis can be repeated for any Surah by changing the Surah title.

3.3.1 Wordcloud analysis

Wordcloud is also useful for doing sentiment analysis by deploying *comparison.cloud()* function. To use this we need to turn the data frame into a matrix with *reshape2*'s *acast()* function. Many steps can all be done with *joins*, *%>%* (piping), and *dplyr* because the data is in *tidy* format.

```
library(reshape2) # to use cast function
tidyESIC %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("#eb52a6", "#54f0b1"),
                   max.words = 200,
                   random.order = FALSE,
                   rot.per = 0.35)
```

Figure 35: Comparison cloud for Saheeh translation

The output for Saheeh is in Figure 35, and for Yusuf Ali is in Figure 36. We can observe clearly the words "fear", "evil", "hell", dominate the negative sentiments; while the words "faith", "righteous", "mercy" dominate the positive sentiments in both Saheeh and Yusuf Ali. The difference between the two sources is probably due to the choice of words used in the translations.

[43] For example, see sentiment narrative in Jane Austen's novels in Silge and Robinson (2017).

Figure 36: Comparison cloud for Yusuf Ali translation

3.4 Statistics of sentiment score

Now we will go deeper into the statistics of sentiment scoring. This exercise will provide two benefits:

a) to understand better how sentiment scores are built upon, and
b) to see how the scores affect the various findings based on the simple visual analysis done in previous sections.

First, let us do a general count of "positive" versus "negative" sentiment words in Saheeh and Yusuf Ali using the *bing* model.

```
ESIC_sent = tidyESIC %>%
  inner_join(get_sentiments("bing")) %>%
  count(sentiment=="positive")
EYAC_sent = tidyEYAC %>%
  inner_join(get_sentiments("bing")) %>%
  count(sentiment=="positive")
sent_ratio_all = data.frame("si_pos" = ESIC_sent$n,
                            "ya_pos" = EYAC_sent$n)
```

Let us tabulate the scores of positive words and negative words and its percentages in both Saheeh and Yusuf Ali.

	Saheeh	Yusuf Ali
Positive words	3,632	4,479
Negative words	4,910	6,129
Percent positive	0.43	0.42
Percent negative	0.57	0.58

A first observation is: "warnings" (negative sentiment) exceeds "glad tidings" (positive sentiment) by about 14 percent consistently in both translations. If we use the ratio of appearance of the word "Basheeran" versus "Nazeeran", which is 0.31 percent, as an indicator of emphasis, then the finding is consistent. This

is to say that Al-Quran emphasizes more on the "warnings" than the "glad tidings". A simple casual first observation is that the words used in the two English translations of the Quran are "biased" towards negative sentiments. (Obviously, our readers will say that this casual finding must be verified with the original Quran in Arabic.)

Let us see the cases for specific conditions, for example, to compare between "Meccan" Surahs and "Medinan" Surahs. If we group the verses according to this category, will the positive over negative sentiments ratio remain consistent with the whole corpus?

```
ESIC_sent_meccan = tidyESIC %>%
  filter(revelation_type=="Meccan") %>%
  inner_join(get_sentiments("bing")) %>%
  count(sentiment=="positive")
EYAC_sent_meccan = tidyEYAC %>%
  inner_join(get_sentiments("bing")) %>%
  filter(revelation_type=="Meccan") %>%
  count(sentiment=="positive")
```

```
ESIC_sent_medinan = tidyESIC %>%
  filter(revelation_type=="Medinan") %>%
  inner_join(get_sentiments("bing")) %>%
  count(sentiment=="positive")
EYAC_sent_medinan = tidyEYAC %>%
  inner_join(get_sentiments("bing")) %>%
  filter(revelation_type=="Medinan") %>%
  count(sentiment=="positive")
```

```
sent_ratio_rev = data.frame("si_pos_meccan" = ESIC_sent_meccan$n,
                            "ya_pos_meccan" = EYAC_sent_meccan$n,
                            "si_pos_medinan" = ESIC_sent_medinan$n,
                            "ya_pos_medinan" = EYAC_sent_medinan$n)
```

Again we tabulate the results below:

	Saheeh	Yusuf Ali
Positive words Meccan	2,173	2,625
Negative words Meccan	2,881	3,775
Percent positive Meccan	0.43	0.41
Percent negative Meccan	0.57	0.59
Positive words Medinan	1,459	1,854
Negative words Medinan	2,029	2,354
Percent positive Medinan	0.42	0.44
Percent negative Medinan	0.58	0.56

Comparing both results, we can observe that the bias towards negative sentiments remains consistent with the whole corpus for both translations. We can say that based on the *bing* sentiment model[44], the messages of "warnings" and "glad tidings" are consistent across revelation periods.[45]

On another note, the number of positive words and negative words in Meccan Surahs (and verses) exceeds the Medinan Surahs; the reason of which is probably attributed to Meccan Surahs having more verses compared to Medinan Surahs.

[44] All works are repeatable with *AFINN* model in a similar manner.

[45] A simple t-test confirms that the statistical distribution of both samples is the same, and another t-test for comparison of sub-samples against the main sample also holds.

3.5 Sentiment scoring frequencies

Sentiment scoring originates from the information theory of signaling. Each word in a sequence of the text reflects a signal, which in sentiment scoring is symbolized by a sign, such as "positive" or "negative", or "neutral" (or other levels of signals). Combined series of signals is considered as messages.

Let us look into the entire corpus of Saheeh and the *bing* sentiment model to visualize the signals.

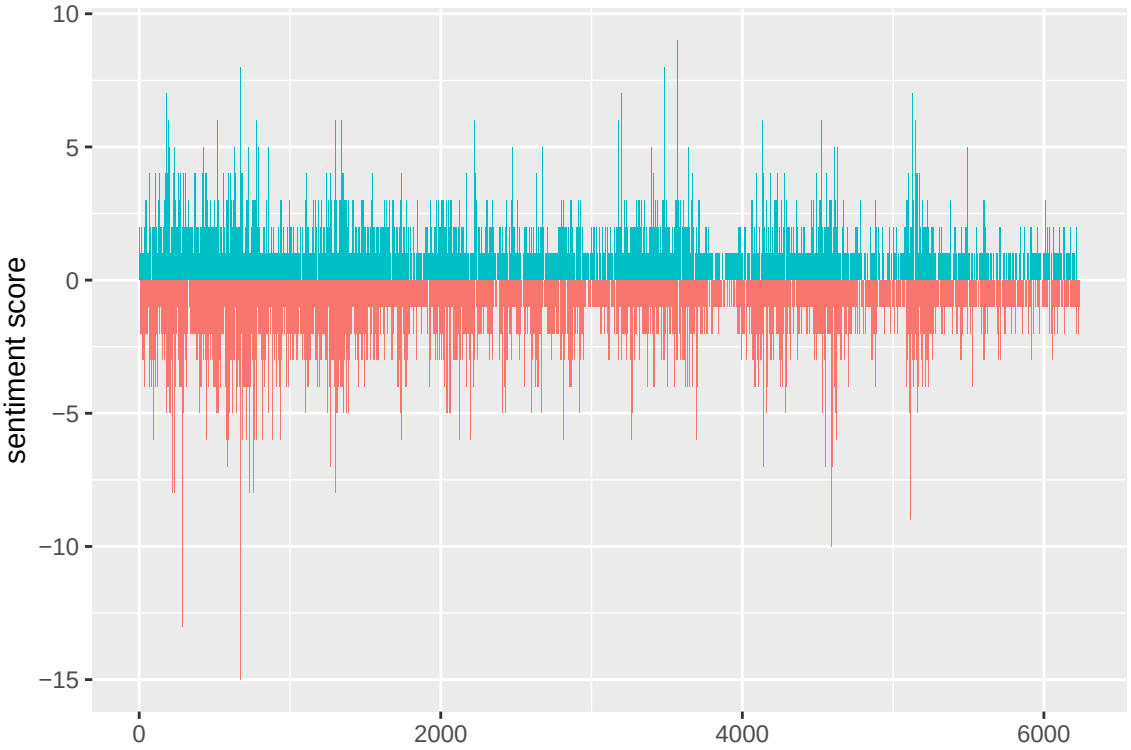


Figure 37: Sentiment frequency in Saheeh corpus

Figure 37 clearly shows the "signals" in terms of frequencies from the first verse till the last verse for the entire Saheeh corpus. There are verses of abnormally high negative signal (largest score of minus 15) and abnormally high positive signal (largest score of plus 10). It is unclear what to make out of the observations unless a deeper analysis is done together with the Sciences of Al-Quran.[46]

The signaling approach of sentiment analysis is also useful to investigate messages within a selected verse or group of verses, such as a Surah. Let us do this for a few selected Surahs: "Yaseen", "Al-Fath", "Al-Mulk", "Al-Muddaththir", "Al-Waaqia", and "Al-Qiyaama". We will choose Saheeh as our source of texts and *bing* as our model.

We chose the selected Surahs for a few reasons. Firstly, they are the Surahs being recited often in daily prayers, in particular Surah Yaseen (for Malaysians), Al-Mulk (for its known virtues). Secondly, they are Surahs with known "strong" messages, such as Surah "Al-Waaqia" and "Al-Qiyaama".

From Figure 38, we can see how these different Surahs "signals" are reflected throughout the Surah. "Yaseen" is balanced between positive and negative. "Al-Fath" (victory) is with many positives and few large negatives. "Al-Waaqia" is with intermittent positives and negatives. "Al-Qiyaama" is dominantly negative, in particular towards the end of the Surah.

The comments we made are in a way, generalized trends of these signals without reference to the content of the Surah and interpretations of these Surahs. Again we leave this as a research question.

[46] We leave this subject as a research question.

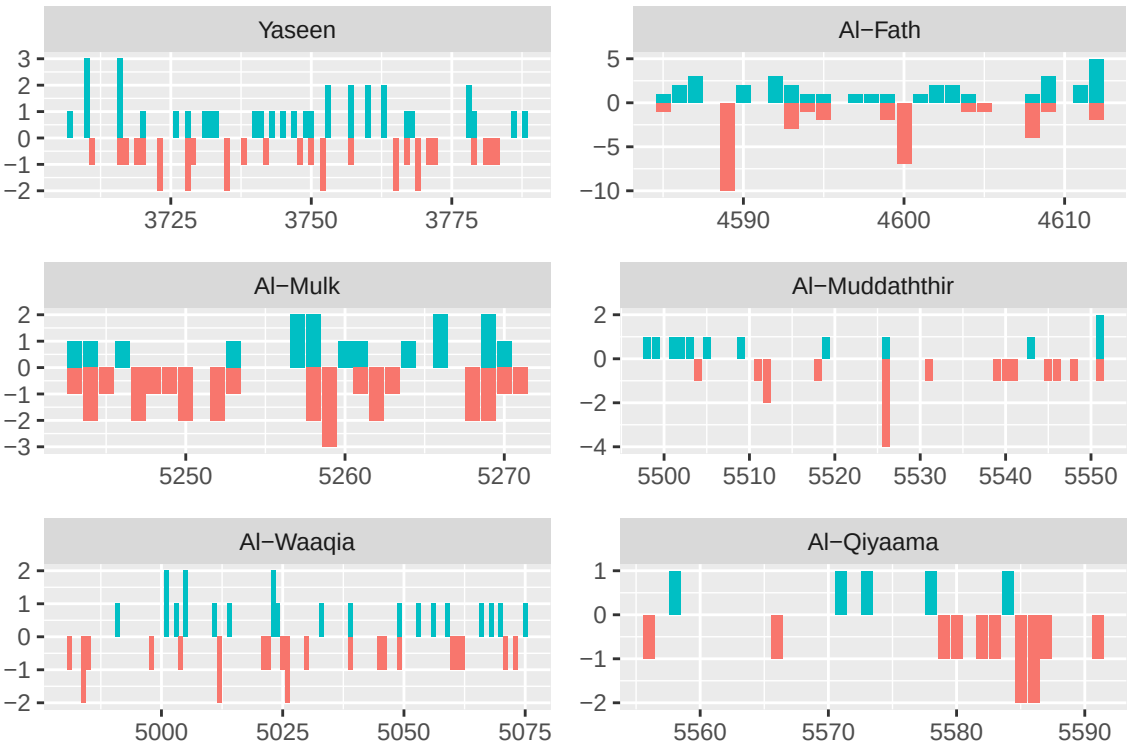


Figure 38: Sentiment frequency in selected Surahs

3.6 Building dedicated sentiment scoring model

Thus far, we have relied on existing pre-built models of sentiment analysis in the English language. Like any model, it is built based on its own sets of assumptions. The performance of these models in the common usage of the English language is quite good and well documented. The question is, do these models match well to analyze the English translations of Al-Quran? It has been argued that any sentiment model is "domain-driven", whereby the training libraries are sourced from a certain domain, such as from the social media (in the case of Twitter sentiment models), or news articles (for a more formal setting), based on American English medium or British English medium (for localization), and so on. Does a simple assumption of using any of the pre-built models applied to the translations of Al-Quran hold?

The task which is important for Quran Analytics is to build dedicated sentiment scoring models based on the Quranic Arabic language. There are few reasons why this is important:

1. The sentiment scoring model for Al-Quran, whether based on the Arabic language or other languages for its translations, must be a dedicated model or models. Firstly, it must be more biased towards the meanings and intents of the messages of Al-Quran. Then it must be less biased for the language, namely Quranic Arabic, Classical Arabic (for Arabic), and any other language medium, such as the English language.
2. Any sentiment scoring model has to rely on Ulum Al-Quran methodologies, and in particular exegesis methodologies, which have been extensively laid out by the Islamic scholars of the past.
3. Furthermore, the dimensions of the "sentiments" should be much larger than a simple binary ("positive" versus "negative") space, and very likely a more complex dimension is necessary.
4. The learning base model, which consists of the corpora, the algorithms of machine learning, inferencing, and the various statistical tools and assumptions must be tested and verified through a rigorous process before deployment in a public setting.

We include the discussion here to highlight the issues and provide views of possible directions and implications

for future research.

3.7 Summary

Sentiment analysis provides a way to understand the attitudes and opinions expressed in texts. In the chapter, we explored how to apply sentiment analysis for two English translations of Al-Quran using tidy data principles. Most of the results are almost similar. What's interesting here is the resounding similarity between the two English translations of the Al Quran, which reflects among other things:

1. The consistencies of methods of translating Al Quran into the English language do not significantly alter the "mood" within the sentences used.
2. Assuming that the two translators might use a different methodology of exegesis and rely on different rules of the English language, yet when a sentiment engine such as *bing* is being applied, we find consistencies in sentiment analysis. This may point to the fact that the original text itself (Al-Quran Arabic) is "rich" in its lexical and semantical meaning.

In closing this chapter, we suggest for further studies to:

1. The expectedly different methods of translating Al Quran into the English language do not significantly alter the "mood" within the sentences used.
2. Assuming that the two translators might use a different methodology of exegesis and rely on different rules of the English language, yet when a sentiment engine such as *bing* is being applied, we find consistencies in sentiment analysis. This may point to the fact that the original text itself (Al-Quran Arabic) is "rich" in its lexical and semantical meaning.

In closing this chapter, we suggest for further research to:

1. Complete a comprehensive study by applying a wider analysis using all available translations of Al-Quran in many languages.
2. Apply various other sentiment scoring analysis models, besides *bing* and *AFINN*, into the analysis.
3. Study the applications of sentiment modeling to the original Quranic texts.

And finally, we also allude to the need of building a Quran Analytics sentiment model which is based on the Sciences of Al-Quran. It must be driven by Quranic Arabic sentiment analysis and extended to languages other than Arabic which have developed their own sentiment models, such as the English language.

3.8 Further readings

Al-Saffar, A., Awang, S., Tao, H., Omar, N., Al-Saiagh, W., and Al-bared, M. *Malay sentiment analysis based on combined classification approaches and senti-lexicon algorithm.* PLOS ONE, 13(4):1–18, 2018. (Al-Saffar et al., 2018)

Dawson, C. W., Ghallab, A., Mohsen, A., and Ali, Y. *Arabic sentiment analysis: A systematic literature review.* Applied Computational Intelligence and Soft Computing, 2020. (Dawson et al., 2020)

Silge, J. and Robinson, D. *Text Mining with R: A Tidy Approach.* O'Reilly Media, Inc., Newton, Massachusetts, 2017. (Silge and Robinson, 2017)

igraph package in **R**. (Csárdi, 2020)

ggraph package in **R**. (Pedersen, 2017)

cleanNLP package in **R**. (Arnold, 2016)

coreNLP package in **R**. (Arnold and Tilton, 2016)

sentimentr package in **R**. (Rinker, 2017)

Analysis of Words by its Cooccurences

"You shall know a word by the company it keeps". "Birds of the same feathers flock together"

Words of common features appear together. Words to other words have relations in linguistic terms. The relations depend on the context, whether in a grammatical sense or sense of meaning. Grammatical rules have their own structures; while meaning also has its own structures, which is further divided as meaning in the sense of grammar and meaning in the sense of semantics.

Locations of words within a sentence, within a group of sentences, within a corpus greatly matters. In particular, for the case of Al-Quran Arabic, the exact location of any word, or for that matter any alphabet, is precisely fixed and determined. In NLP, the relationship between words is a major undertaking, with recognized advancements of its own in certain languages such as the English language. In Quran Analytics, we consider this as a subject of its own. Since we are focused on analyzing the translations of Al-Quran, we will deal with the subject using the translated corpora of Al-Quran.

As an introduction to the subject, in the next three chapters, we will address the subject of the relationship between words from three different perspectives. The first perspective is from the locations of words with other words in the corpus, or "collocations" and "co-occurrences" (Chapter 4). The second perspective is to look at these relations from statistical aspects, namely the correlations and other statistical measures (Chapter 5). The third perspective is to look at the relationship from a holistic or global approach, namely by looking at all of the words as a network of words (Chapter 6)).

The three chapters will be introductory materials for the subject of roles of words within a much wider and complex subject. There are many advancements made along these lines in recent studies of Computational Linguistics which we will not cover in this book. We will leave these issues as notes for further research.

4 Word Collocations

The word is among the most elementary units of language. It forms the vocabulary, is part of the grammatical structures, builds the semantics of a sentence, and provides many contexts of usage. Among the important tools developed in NLP is to model the relationships between words, in a sentence, in a corpus, and across corpora of text collection. Sentences or speeches are organized as sequences of words, upon which the concept of word relations is built.

In this chapter, we will cover the broad ideas on the role of words forming the building blocks of grammatical and lexical structures of the sentence, with a particular focus on word collocations.

Al-Quran, as a preserved text, is precise in its word arrangements, from the first verse to the last. Nothing has changed from the beginning. Nothing can be altered. The Quranic structure of the words is an integral part of the Quran. When translated into another language, the contextual meaning of the translated text must follow the structures of the original text to a large degree; therefore the word relationship structure should be preserved as much as possible within the translations.

In this chapter, we will explore some of the analysis on the relationship between words in the English translations of Al-Quran, namely the Saheeh and Yusuf Ali, as was the approach in the previous chapters. We explore some of the methods *tidytext* (Queiroz et al., 2020) package offers for calculating and visualizing relationships between words in these translations. The analysis includes understanding the structure of words occurring together (called n-grams), which words tend to follow another word (word correlations), how various words relate to each other (word network), and words appearance within sentences or a selected group of texts (such as a Surah or Hizb). Furthermore, we will introduce methods of lexical analysis whereby all the processes of lexical annotation, tagging, and lemmatization are applied to Surah Yusuf as a sample analysis.

We will resume using the same environment in Chapter 3, whereby all the data used is the same and all the **R** libraries remain intact. We will also use a few new packages: *igraph* (Csárdi, 2020), *ggraph* (Pedersen,

2017), which extends *ggplot2* to construct network plots, and *widyr* (Robinson et al., 2020), which calculates pairwise correlations and distances within a tidy data frame. For the lexical analysis, we will deploy the *udpipe* (Wijffels et al., 2020) package, which includes the Udpipe[47] language model pre-loaded into the computing environment.

4.1 Analyzing word collocations

The simplest relationship between words in sentences is "its neighbour(s)", word(s) preceding it, and word(s) after it. This relation is called word collocations. In NLP this is calculated using *n-grams*. In probabilistic terms, what we want to guess is, if a word "x" is known, what is the most probable word "y" that will follow, and vice-versa if we know "y", what is the most probable word "x" that precedes it. This is called a *bi-gram* (or two words sequence) and can be extended to three words sequence, a *tri-gram*, and so on. If we have 3,000 distinct words (tokens), given a word, there are 2,999 possible choices of adjacent words, which means the probability space is not only large but also sparse. If we extend the same logic to tri-grams, the space increases exponentially. To create a model of n-grams, a probability model must be deployed.[48]

4.1.1 Analyzing bi-grams

In **R** we can use the *tidytext* package and apply the tokenization function with the addition of *token = "ngrams"* option to *unnest_tokens()* function. When we set $n = 2$, we are examining pairs of two consecutive words, which is called "bi-grams". We show this below:

```
ESI_bigrams <- quran_all %>%
  unnest_tokens(bigram, saheeh, token = "ngrams", n = 2)
```

This data structure is still a variation of the *tidytext* format. It is structured as one-token-per-row (with extra metadata, such as surah, still preserved), but each token now represents a bigram. Notice that these bigrams overlap: "in the", "the name", "name of" are separate bigrams.

We can examine the most common bigrams using dplyr's *count()*:

```
ESI_bigrams %>%
  count(bigram, sort = TRUE)
```

Many of the most common bigrams are pairs of common (uninteresting) words, such as "of the", "those who", "and the", "do not". We call these "stopwords". We use tidyr's *separate()* function, which splits a column into multiple columns based on a delimiter. This lets us separate it into two columns, "word1" and "word2", at which point we can remove cases where either is a stopword.

```
ESI_bigrams_separated <-  ESI_bigrams %>%
  separate(bigram, c("word1", "word2"), sep = " ")

ESI_bigrams_filtered <- ESI_bigrams_separated %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)

ESI_bigram_counts <- ESI_bigrams_filtered %>%
  count(word1, word2, sort = TRUE)
```

In other analyses, we may want to work with the recombined words. *tidyr*'s *unite()* function is the inverse of *separate()* and lets us recombine the columns into one. Thus, "separate/filter/count/unite" let us find the most common bi-grams not containing the stopwords.

[47] Language model developed by the Institute of Formal and Applied Linguistics, Charles University, Czech republic.
[48] A comprehensive textbook on n-gram models is (Manning and Schutze, 1999).

```
ESI_bigrams_united <- ESI_bigrams_filtered %>%
  unite(bigram, word1, word2, sep = " ")
```

This one-bigram-per-row format is helpful for exploratory analyses of the text. As a simple example, we might be interested in the most common word with "allah" mentioned in each Surah. The result is presented in Figure 39.

```
ESI_bigrams_filtered %>%
        filter(word1 == "allah") %>%
        count(surah_title_en, word2, sort = TRUE)
```

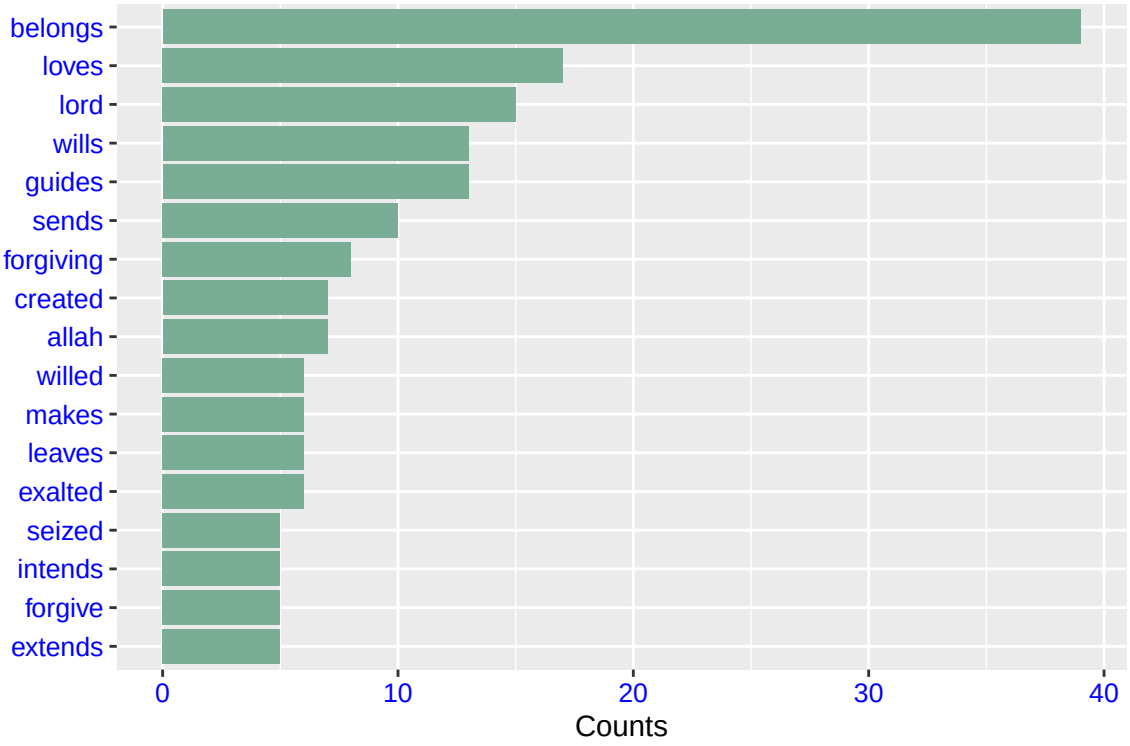


Figure 39: Top words with 'Allah' in the Quran

A bigram can also be treated as a term in a document in the same way that we treated individual words. For example, we can look at the tf-idf of bigrams across Surahs. These tf-idf values can be visualized within each long surah, just as we did for words in Chapter 2. This is shown in Figure 40.

There are advantages and disadvantages to examining the tf-idf of bigrams rather than individual words. Pairs of consecutive words might capture structure that isn't present when we are just counting single words and may provide context that makes tokens more understandable (for example, "sacred house", in Al-Maaida, is more informative than "house" or "sacred" separately). However, the per-bigram counts are also more sparse: a typical two-word pair is rarer than either of its component words. Thus, bigrams can be especially useful when you have a very large text dataset.

4.1.2 Visualizing a network of bigrams with *ggraph*

We may be interested in visualizing all of the relationships among words simultaneously, rather than just the top few at a time. One common visualization method is to arrange the words in a network graph. Here we will be referring to a "graph" as a combination of connected nodes. A graph can be constructed from a tidy object since it has three variables:

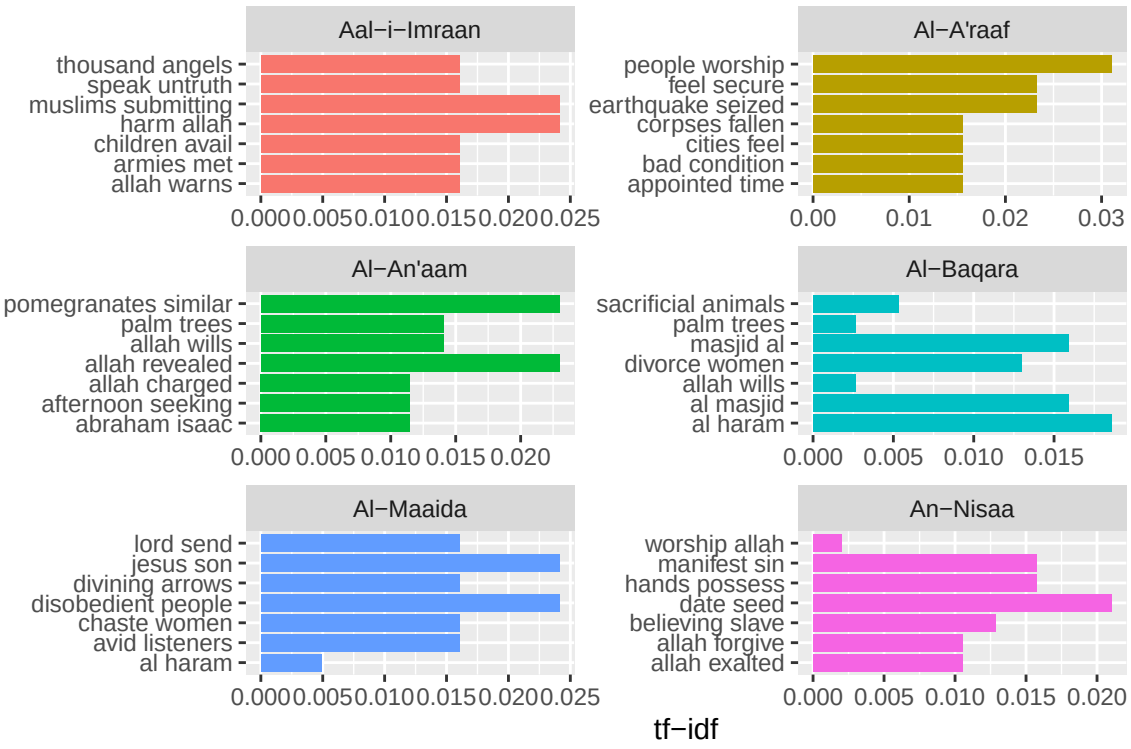


Figure 40: Top word pairs tf-idf in the long Surahs

1. *from*: the node an edge is coming from
2. *to*: the node an edge is going towards
3. *weight*: A numeric value associated with each edge

The *igraph* package has many powerful functions for manipulating and analyzing networks. One way to create an *igraph* object from tidy data is the *graph_from_data_frame()* function, which takes a data frame of edges with columns for "from" (word1), "to" (word2), and edge attributes (in this case n).

```
ESI_bigram_graph <- ESI_bigram_counts %>%
                    filter(n >= 10) %>%
                    graph_from_data_frame()
```

igraph has built-in plotting functions, but many other packages have developed better visualization methods for graph objects. The *ggraph* package (Pedersen, 2017) implements these visualizations in terms of the grammar of graphics. We can convert an *igraph* object into a *ggraph* with the *ggraph* function, after which we add layers to it, much like adding layers in *ggplot2*. For example, for a basic graph, we need to add three layers: nodes, edges, and text.

Figure 41 shows all the top bi-grams (count above 10) for Saheeh's translation. From here we can make some observations, like the word "allah" having a dominant role, and some known concepts in Islam like "straight path" and "establish prayer". "perpetual residence", "rivers flow", "gardens beneath" are known rewards for those who "enter paradise".

We conclude with some polishing steps to make a nicer plot and at the same time reflect the attributes within the plot, such as re-sizing the edges to reflect the weights of the relations. The codes are presented below and the resulting plot is in Figure 42.

It may take some experimentation with ggraph to get your networks into a presentable format, but network visualization is a useful and flexible way to look at relational tidy data.

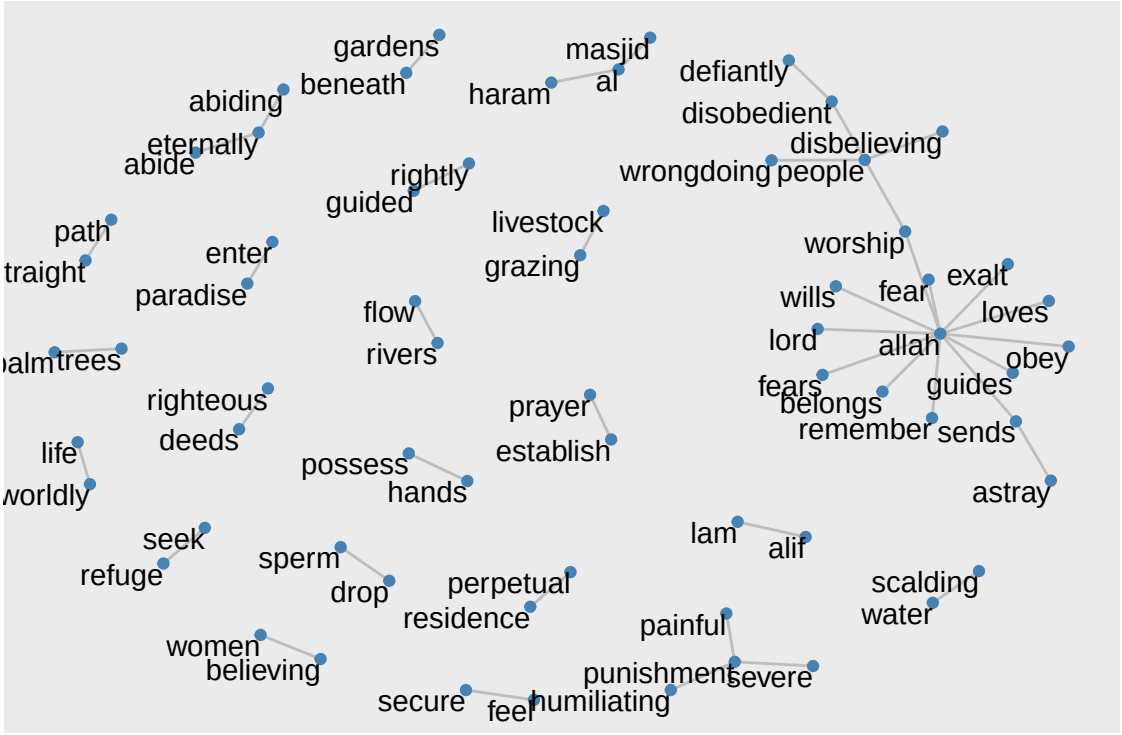


Figure 41: Common bi-grams in Saheeh

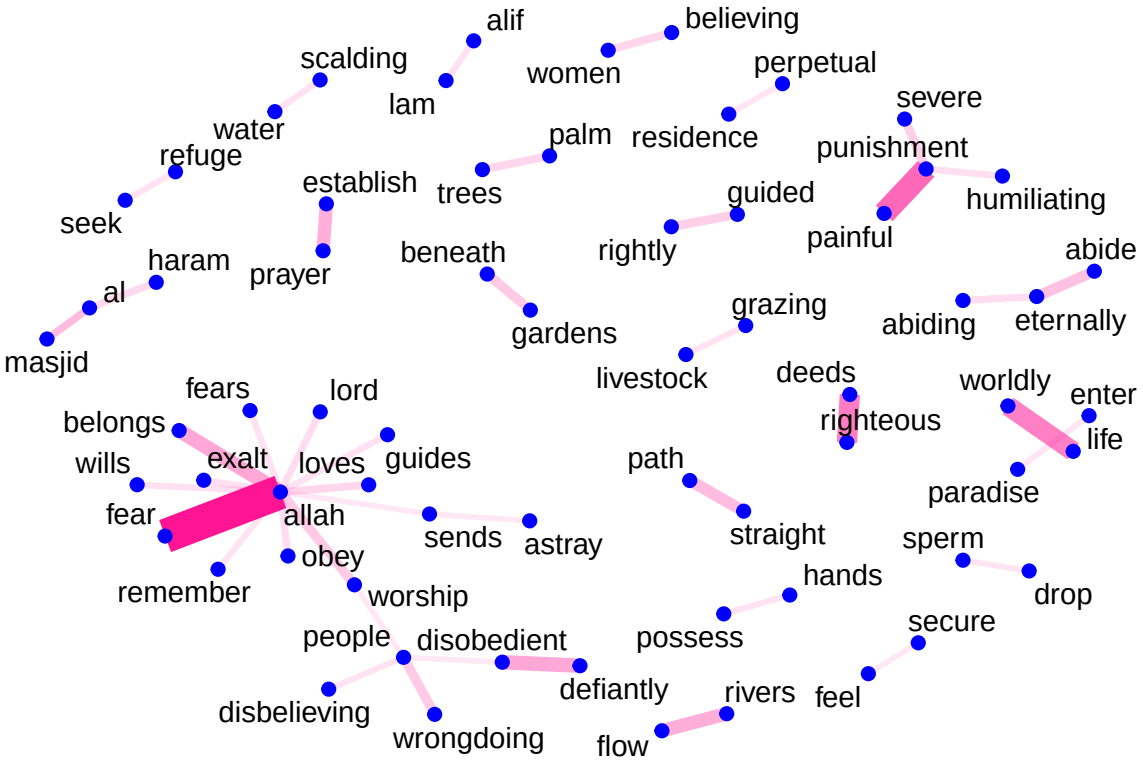


Figure 42: Common bi-grams in Saheeh with ggraph format

Note that this is a visualization of a Markov chain,[49] a common model in text processing. In a Markov chain, each choice of a word depends only on the previous word. In this case, a random generator following this model might spit out "lord", then "loves", then "guides/obey", by following each word with the most common words that follow it. To make the visualization interpretable, we chose to show only the most common word-to-word connections, but one can imagine an enormous graph representing all connections that occur in the Quran.

4.1.3 Tri-grams

We discuss next the most common trigrams, which are consecutive sequences of 3 words. We can find this by setting $n = 3$ as shown in the code below:

```
ESI_trigram_counts <- quran_all %>%
  unnest_tokens(trigram, saheeh, token = "ngrams", n = 3) %>%
  separate(trigram, c("word1", "word2", "word3"), sep = " ") %>%
  filter(!word1 %in% stop_words$word,
         !word2 %in% stop_words$word,
         !word3 %in% stop_words$word) %>%
  count(word1, word2, word3, sort = TRUE)
ESI_trigram_counts <- ESI_trigram_counts %>%
                          filter(!is.na(word1) & !is.na(word2) & !is.na(word3))
ESI_trigrams_united <- ESI_trigram_counts %>%
                          unite(trigram, word1, word2, word3, sep = " ")
```

The result is viewed with:

```
head(ESI_trigrams_united)
```

Similar analyses performed for bigrams may be repeated for trigrams as well. We leave this subject for readers to try.

4.1.4 Bigrams co-ocurrences and correlations

As a next step, let us examine which words commonly occur together in the Surahs. We can then examine word networks for these fields; this may help us see, for example, which Surahs are related to each other.

We can use *pairwise_count()* from the *widyr* package to count how many times each pair of words occur together in a Surah.

```
word_pairs <- surah_wordsESI %>%
  pairwise_count(word, surah_title_en, sort = TRUE, upper = FALSE)
head(word_pairs)
```

Let us create a graph network of these co-occurring words so we can see the relationships better. The filter will determine the size of the graph.

Figure 43 exhibits the words (nodes) and their links (edges) to their pairs (other nodes) in the word pairs network for Saheeh. The size of the nodes indicates the degree of connections it has. The thickness of the links indicates the frequencies of links. At the center is the word "allah", "lord" as expected. However, we can see many other words (nodes) such as "prayer", which is highly linked to the center words (such as "allah"). Network visualization is a good starting point to make these types of observations.

[49] A Markov chain is a mathematical system that experiences transitions from one state to another according to certain probabilistic rules. The defining characteristic of a Markov chain is that no matter how the process arrived at its present state, the possible future states are fixed. https://brilliant.org/wiki/markov-chains/

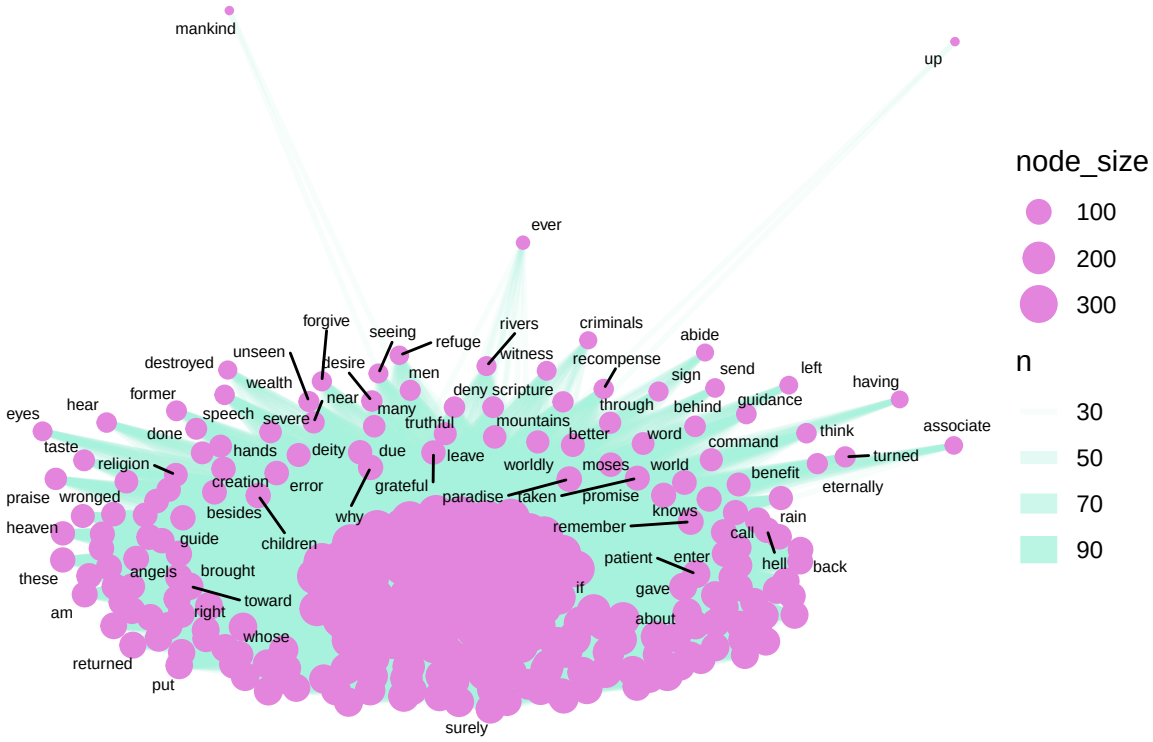


Figure 43: Common word pairs in Saheeh

4.1.5 Visualizing correlations of bigrams of keywords

Now we examine the relationships among keywords in a different way. We can find the correlation among the keywords by looking for those keywords that are more likely to occur together than with other keywords for a dataset.

```
keyword_cors <- surah_wordsESI %>%
  group_by(word) %>%
  filter(n() >= 50) %>%
  pairwise_cor(word, surah_title_en, sort = TRUE, upper = FALSE)
```

Let us visualize the network of keyword correlations.

From Figure 44, we can visualize all the keywords and their relations with other keywords. Note that the word “allah” and “lord” while being major keywords are less correlated to each other and other words. It shows that major keywords do not necessarily have high correlations with other keywords. This is the idea behind “keyword in context” (kwic) which we will cover in a later part of the book.

4.1.6 Summarizing ngrams

We have shown some methods of creating bigrams and analyzing them. Bigrams are the simplest relations between words, and the statistical properties of bigrams are rather straightforward. However, once we start to add more than two words, the complications grow exponentially. Let’s say that we have a window of five words (5-grams); what we have now is a space of four preceding words and four succeeding words, where the window is a moving window. Every time we move one word ahead, we move along the window and calculate the relations four steps backward and four steps forward. This is a very tedious and compute-heavy exercise. The same analyses done on bigrams explode into bigger scale computations when extended to a larger number of words and visual analyses are no longer possible.

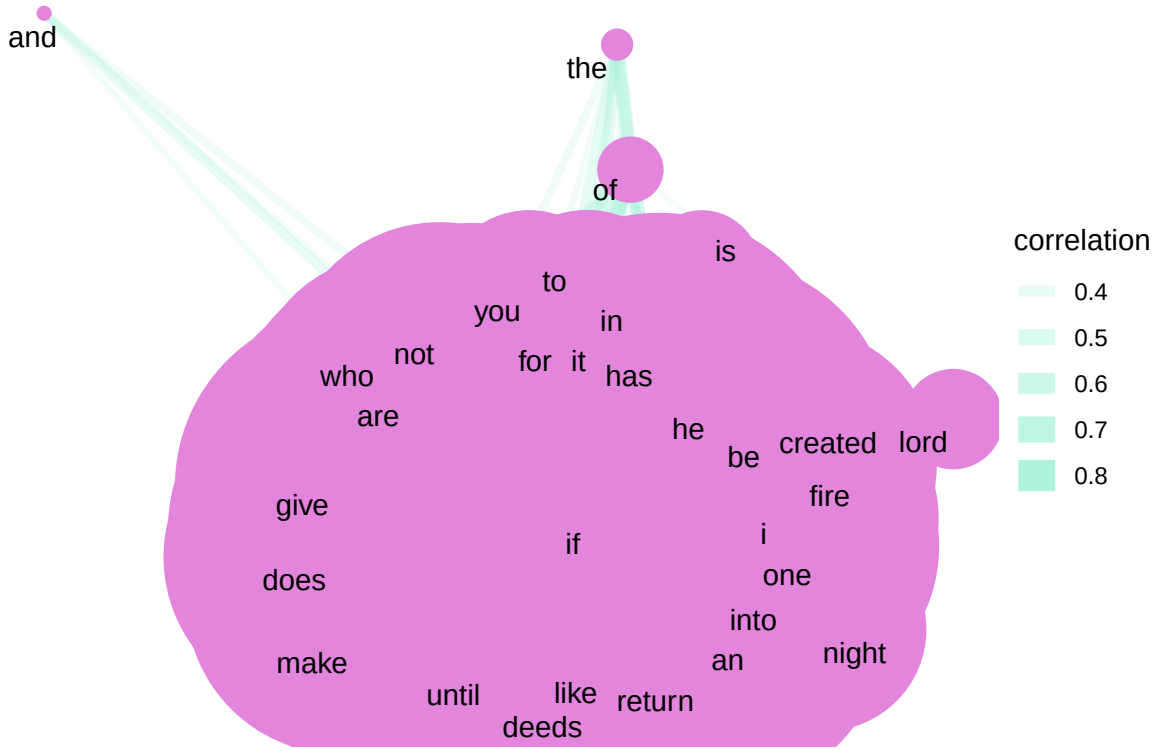


Figure 44: Common keyword correlations in Saheeh

Besides n-grams, we also have "skip-grams". In skip-grams, instead of looking at words next to each other, we look at words within a distance to each other, such as three words away. Skip grams have their own purpose and use in analytics.

There are powerful algorithms used in NLP which utilize fast-speed mechanisms to convert words into compact vector representations. The development of these algorithms is to overcome computer memory problems, where when we have a large number of words in big corpora (such as the Internet), computing n-grams, skip-grams, will take too much memory space, and hence require compact space representations. This is especially when we want to apply learning models in Machine Learning algorithms such as Neural Network models.

4.2 Lexical analysis

An important NLP task in describing the relationships between words is lexical analysis. There are many good introductions to the subject, such as in Manning and Schutze (1999) and Manning et al. (2009). The first step for lexical analysis involves Part-of-Speech (POS) tagging, stemming, and lemmatization.[50] This is followed by syntactic parsing, which is the identification of words within grammatical rules (such as a verb, a noun, a phrase, etc.). Finally, the step involves dependency parsing, which is the process of analyzing the grammatical structure of sentences (i.e., the sequence of words in grammatical rules).

For lexical analysis processing, we deploy the *udpipe* package in **R** which was developed by Wijffels et al. (2020).[51] For the purpose of demonstration, we select Surah Yusuf as our sample of analysis. The Surah is a fairly long chapter with 111 verses and mainly narrates the story of Prophet Yusuf (Joseph), which makes the analysis interesting.

[50]The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

[51]The udpipe model is developed by the Institute of Formal Applied Linguistics, Charles University, Czech republic. Guides for using the package are available at (https://bnosac.github.io/udpipe/docs/doc5.html)

We start with loading the *udpipe* language model for the English language, using the *udpipe_download_model()* function.

```
# During first time model download execute the below line too
# library(udpipe)
# udmodel <- udpipe_download_model(language = "english")
# Load the model
udmodel <- udpipe_load_model(file = 'data/english-ewt-ud-2.5-191206.udpipe')
```

We will start by annotating Surah Yusuf. The annotated *data.frame* is used for the analysis later.

```
Q01 <- quran_en_sahih %>% filter(surah == 12)
x <- udpipe_annotate(udmodel, x = Q01$text, doc_id = Q01$ayah_title)
x <- as.data.frame(x)
```

The resulting *data.frame* has a field called upos which is the Universal Parts of Speech tag and also a field called lemma which is the root form of each token in the text. These two fields give us a broad range of analytic possibilities.

4.2.1 Basic frequency statistics

In most languages, nouns (NOUN) are the most common types of words, next to verbs (VERB). These are the most relevant for analytic purposes, next to the adjectives (ADJ) and proper nouns (PROPN).[52] The results of the frequencies are shown in Figure 45.

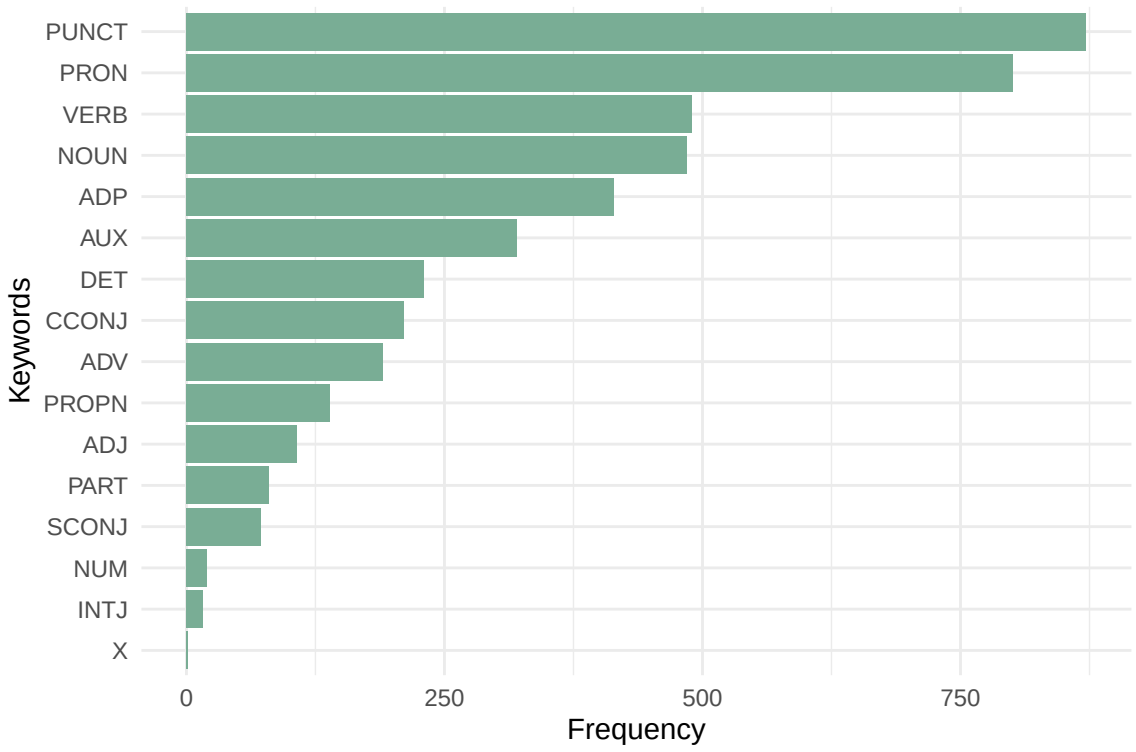


Figure 45: UPOS (Universal Parts of Speech) in Surah Yusuf

Parts of Speech (POS) tags allow us to extract easily the words we like to plot. We may not need stopwords for doing this, we just select nouns or verbs or adjectives and we have the most relevant parts for basic frequency analysis.

[52]For a detailed list of all POS tags, please visit https://universaldependencies.org/u/pos/index.html

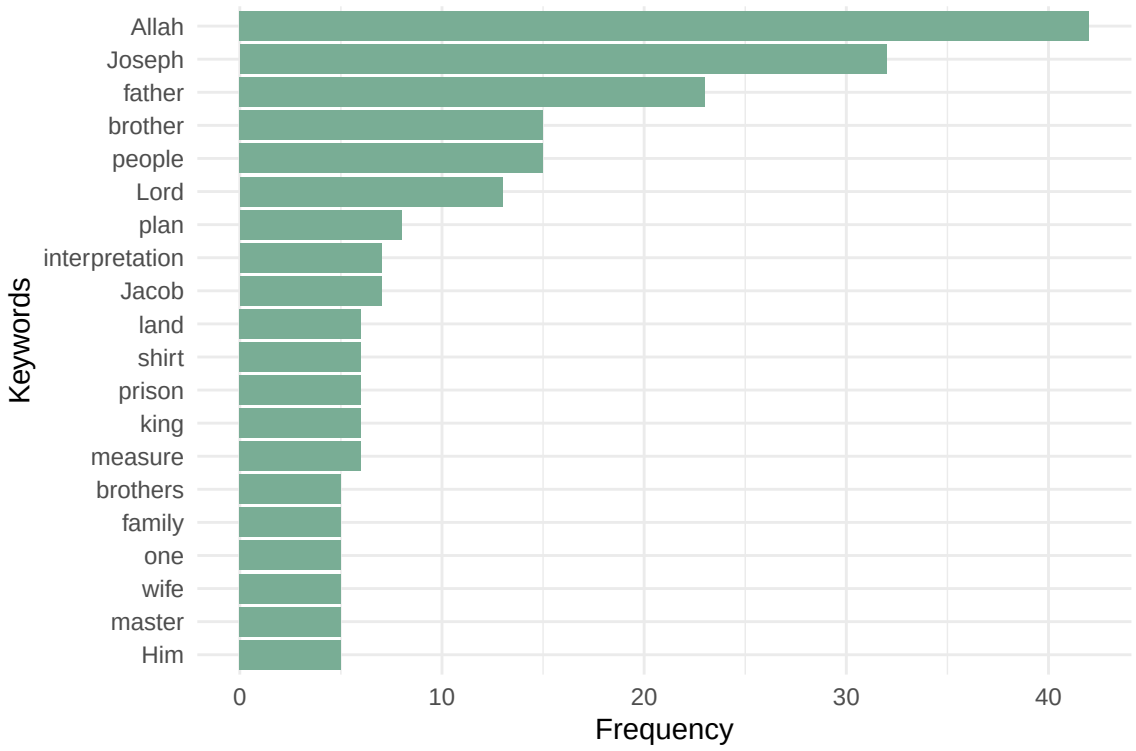


Figure 46: Most occurring nouns in Surah Yusuf

The NOUN and PROPN frequency plot in Figure 46 correctly reflects Allah (SWT) (also the words Lord, Him) as the central dominant subject matter of the Quran (Alsuwaidan and Hussin, 2021). The noticeable noun missing in the plot is "prison". The others are all recognizable to those familiar with Surah Yusuf. These are shown in Figure 47 and Figure 48.

4.3 Word cooccurrences using POS

Analyzing single words is a good start. Multi-word expressions should be more interesting. We can get multi-word expressions by looking either at collocations (words following one another), at word co-occurrences within each sentence, or at word co-occurrences of words that are close in the neighborhood of one another.

Co-occurrences allow us to see how words are used either in the same sentence or next to each other. The udpipe package easily helps us create word co-occurrence graphs using the relevant POS tags.

4.3.1 Nouns, adjectives, and verbs used in same sentence

We look at how many times nouns, proper nouns, adjectives, verbs, and adverbs are used in the same sentence.

```
cooccur <- cooccurrence(x = subset(x, upos %in% c("NOUN", "PROPN", "VERB", "ADJ")),
                        term = "lemma",
                        group = c("doc_id", "paragraph_id", "sentence_id"))
head(cooccur)
```

The result can be easily visualized using the *igraph* and *ggraph* packages. This is shown in Figure 49.

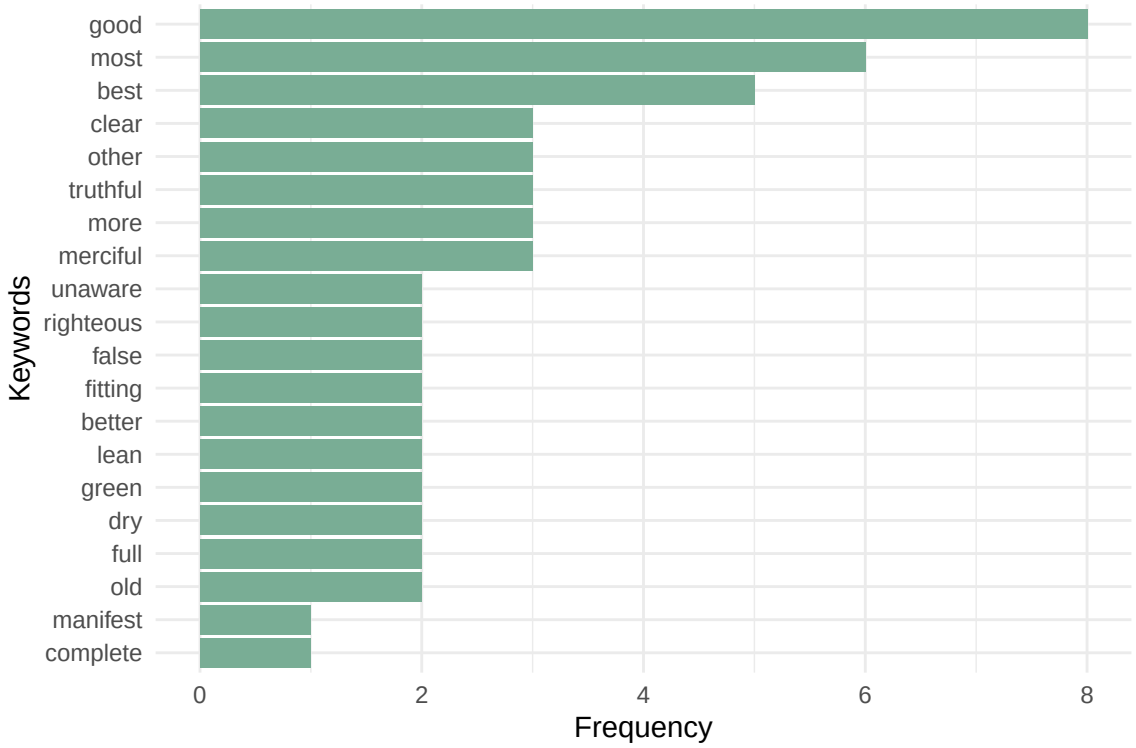


Figure 47: Most occuring adjectives in Surah Yusuf

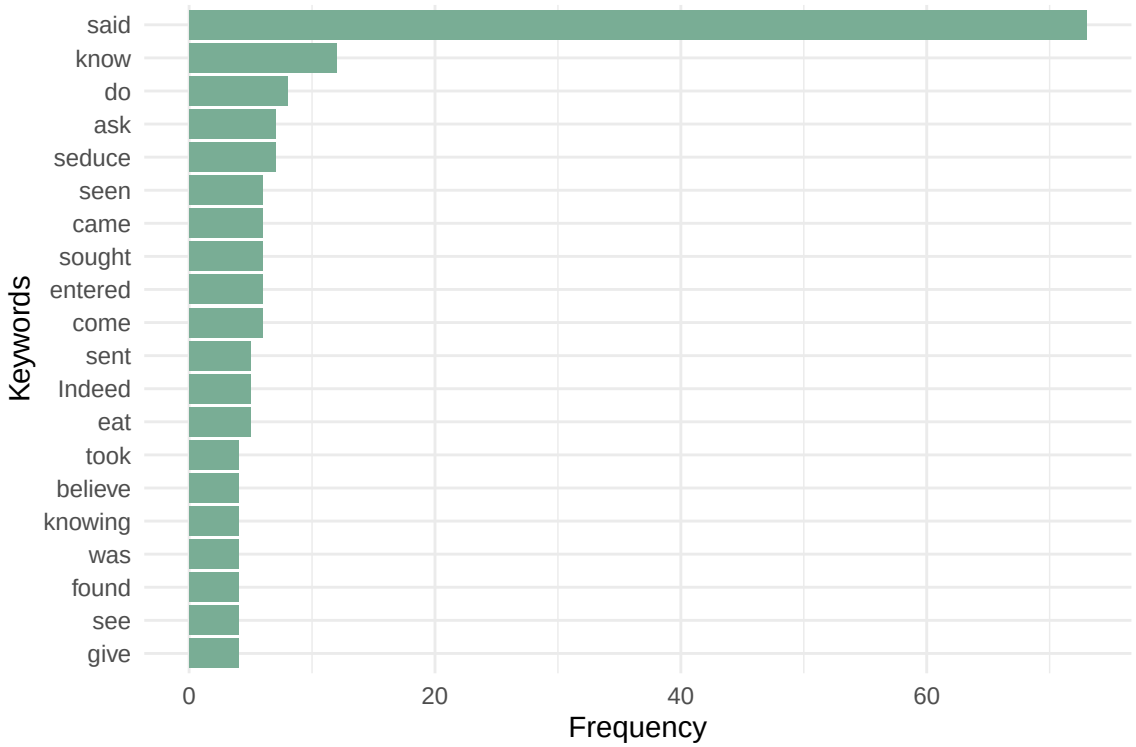


Figure 48: Most occuring verbs in Surah Yusuf

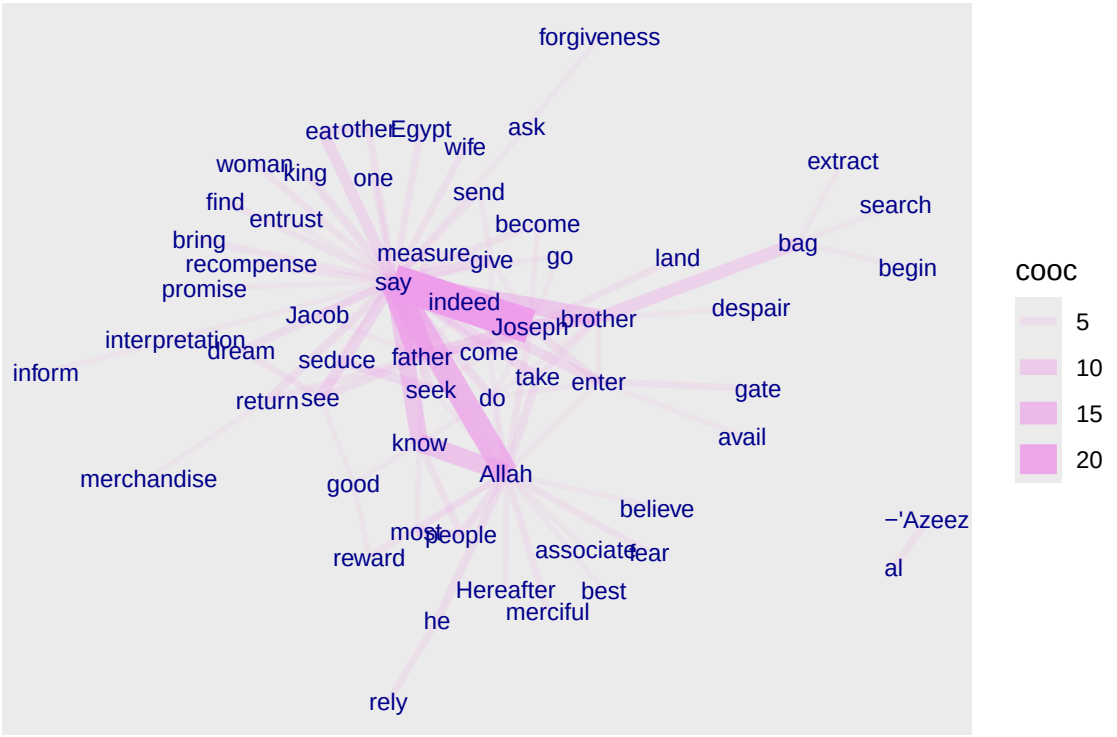


Figure 49: Co-occurence of Nouns, Names, Adjectives and Verbs

The story is revealed by Allah (SWT). The main characters are Joseph, his father, his brothers, the king, and the wife of the minister (al-'Azeez). So the verb "say" dominates since it is a narrated story. It is interesting to note the strong link and occurrence of "know" with "Allah".

4.3.2 Words that follow one another using POS

Visualizing which words follow one another (bigram) can be done by calculating word co-occurrences of a specific POS type that follow one another and specify how far away we want to look regarding "following one another" (in the example below we indicate skipgram = 1 which means look to the next word and the word after that). Here we include the major POS.

```
cooccur <- cooccurrence(x$lemma,
              relevant = x$upos %in% c("NOUN", "PROPN", "VERB", "ADV", "ADJ"),
              skipgram = 1)
```

Once we have these co-occurrences, we can easily do the same plots as before. (See Figure 50).

4.3.3 Word correlations using POS

Keyword correlations indicate how terms are just together in the same document/sentence. While co-occurrences focus on frequency, correlation measures between 2 terms can also be high even if 2 terms occur only a small number of times but always appear together.

We show how nouns, proper nouns, verbs, adverbs, and adjectives are correlated within each verse of Surah Yusuf.

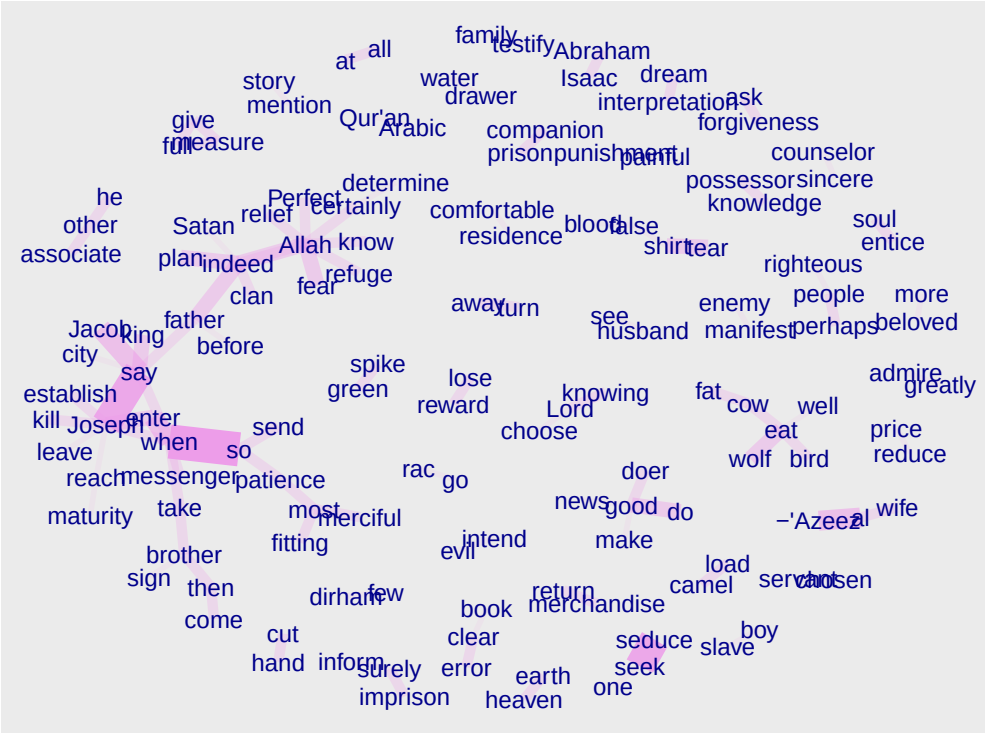


Figure 50: Words following one another in Surah Yusuf

```
x$id <- unique_identifier(x, fields = c("sentence_id", "doc_id"))
dtm <- subset(x, upos %in% c("NOUN", "PROPN", "VERB", "ADV", "ADJ"))
dtm <- document_term_frequencies(dtm, document = "id", term = "lemma")
dtm <- document_term_matrix(dtm)
dtm <- dtm_remove_lowfreq(dtm, minfreq = 5)
termcorrelations <- dtm_cor(dtm)
y <- as_cooccurrence(termcorrelations)
y <- subset(y, term1 < term2 & abs(cooc) > 0.2)
y <- y[order(abs(y$cooc), decreasing = TRUE), ]
head(y)
```

The above pairings indeed reflect the story of Prophet Joseph.

4.4 Finding keyword combinations using POS

Frequency statistics of words are nice but many words only make sense in combination with other words. Thus we want to find keywords that are a combination of words. The steps here follow the example from *An overview of keyword extraction techniques.*[53] The steps are as follows:

1. by doing Parts of Speech tagging to identify nouns
2. based on Collocations and Co-occurrences
3. based on RAKE (Rapid Automatic Keyword Extraction)
4. by looking for Phrases (noun phrases/verb phrases)
5. based on the Textrank algorithm
6. based on results of dependency parsing (getting the subject of the text)

[53] https://www.r-bloggers.com/2018/04/an-overview-of-keyword-extraction-techniques/

Currently, the *udpipe* package provides three methods to identify keywords in text:

1. RAKE (Rapid Automatic Keyword Extraction)
2. Collocation ordering using Pointwise Mutual Information
3. Parts of Speech phrase sequence detection

4.4.1 Using RAKE

RAKE is one of the most popular (unsupervised machine learning) algorithms for extracting keywords. It is a domain-independent keyword extraction algorithm that tries to determine key phrases in a body of text by analyzing the frequency of word appearance and its co-occurrence with other words in the text.

RAKE looks for keywords by looking to a contiguous sequence of words that do not contain irrelevant words by calculating a score for each word that is part of any candidate keyword. This is done by

- among the words of the candidate keywords, the algorithm looks at how many times each word is occurring and how many times it co-occurs with other words
- each word gets a score which is the ratio of the word degree (how many times it co-occurs with other words) to the word frequency

A RAKE score for the full candidate keyword is calculated by summing up the scores of each of the words which define the candidate keyword. The result is in Figure 51.

```
stats <- keywords_rake(x = x,
            term = "lemma",
            group = "doc_id",
            relevant = x$upos %in% c("NOUN", "PROPN", "VERB", "ADV", "ADJ"))
head(stats)
head(stats)
```

4.4.2 Using Pointwise Mutual Information Collocations

The result is in Figure 52.

```
x$word <- tolower(x$token)
stats <- keywords_collocation(x = x, term = "word", group = "doc_id")
stats$key <- factor(stats$keyword, levels = rev(stats$keyword))
head(stats)
```

4.4.3 Using a sequence of POS tags (noun phrases)

Another option is to extract phrases. These are defined as a sequence of POS tags. Common types of phrases are noun phrases or verb phrases. In English, a noun and a verb can form a phrase like "Joseph said". With the noun Joseph and verb said, we can understand the context of the sentence. We will show an example of how to extract noun phrases using the *as_phrasemachine()* function in *udpipe*.

POS tags are re-coded to one of the following one-letters:

- A: adjective
- C: coordinating conjunction
- D: determiner
- M: modifier of a verb

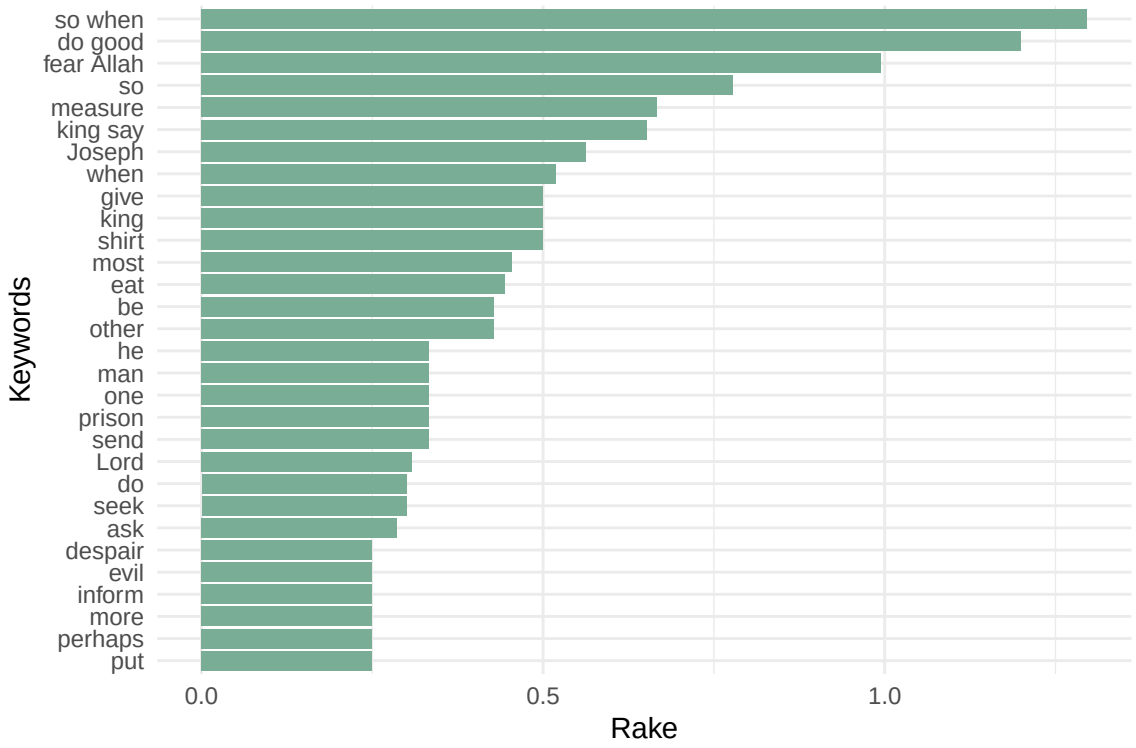


Figure 51: Keywords identified by RAKE in Surah Yusuf

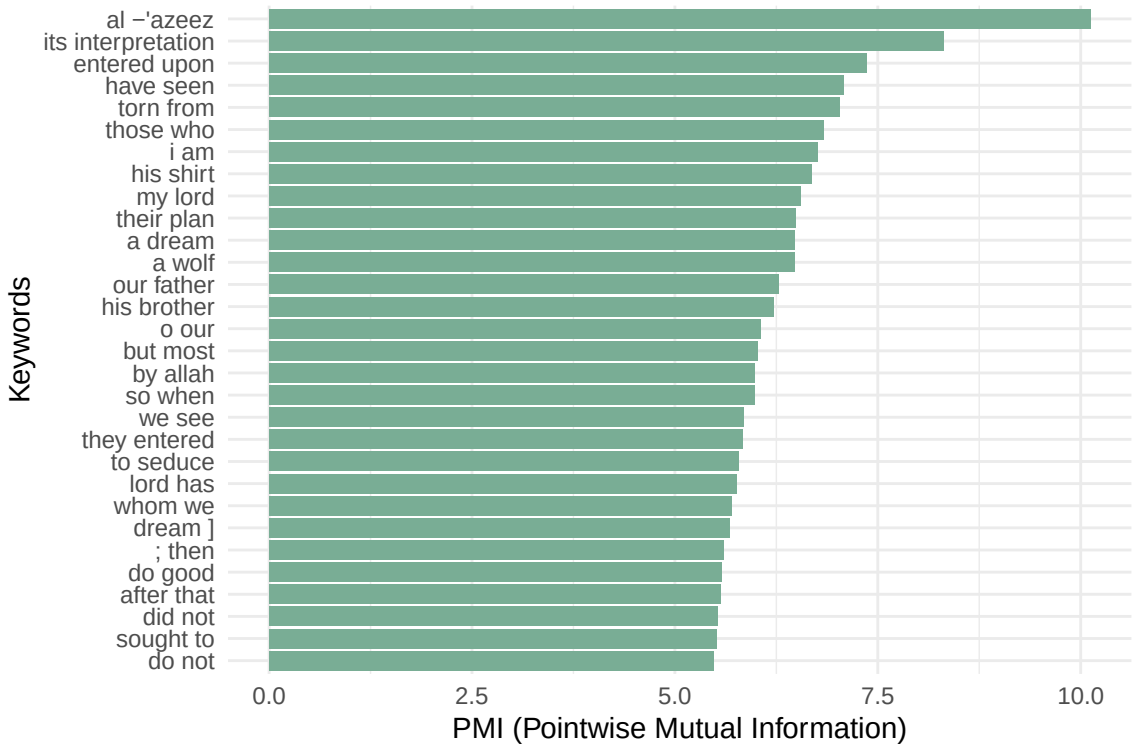


Figure 52: Keywords identified by PMI Collocation in Surah Yusuf

- N: noun or proper noun
- P: preposition

This simple example maps the various UPOS re-coded using letters.

```
y <- c("PROPN", "SCONJ", "ADJ", "NOUN", "VERB", "INTJ", "DET", "AUX", "NUM", "X", "PRON", "PUNCT", "ADP", "CCONJ")
as_phrasemachine(y)
```

We then define a regular expression to indicate a sequence of POS tags which we want to extract from the text. Extracting noun phrases from a text can be done easily by defining a sequence of UPOS tags. For example, this sequence of UPOS tags can be seen as a noun phrase: Adjective, Noun, Preposition, Noun. After which identifying a simple noun phrase can be just expressed by using the following regular expression (A|N)*N(P+D*(A|N)*N)* which says start with adjective or noun, another noun, a preposition, determiner adjective or noun and next to a noun again.[54] The result is in Figure 53.

```
x$phrase_tag <- as_phrasemachine(x$upos, type = "upos")
stats <- keywords_phrases(x = x$phrase_tag, term = tolower(x$token),
                          pattern = "(A|N)*N(P+D*(A|N)*N)*",
                          is_regex = TRUE, detailed = FALSE)
stats <- subset(stats, ngram > 1 & freq > 3)
stats$key <- factor(stats$keyword, levels = rev(stats$keyword))
head(stats)
```

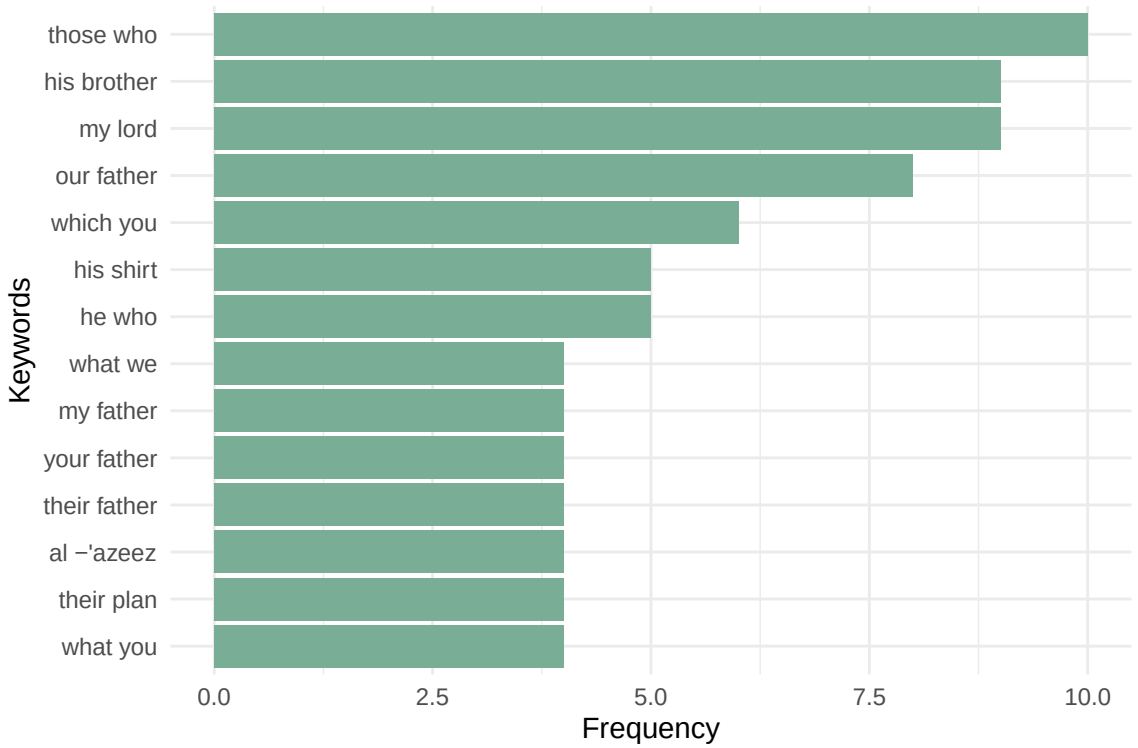


Figure 53: Keywords - simple noun phrases in Surah Yusuf

[54] https://www.rdocumentation.org/packages/udpipe/versions/0.8.5/topics/keywords_phrases

4.4.4 Textrank

Textrank is a word network ordered by Google Pagerank as implemented in the *textrank* (Wijffels and BNOSAC, 2020) package.[55] The algorithm allows to summarize text and extract keywords. This is done by constructing a word network by looking if words are following one another. On top of that network, the 'Google Pagerank' algorithm is applied to extract relevant words after which other relevant words which are following one another are combined to get keywords. In the example below, we are interested in finding keywords using that algorithm of either "NOUN", "PROPN", "VERB", "ADJ" following one another.

```
stats <- textrank::textrank_keywords(x$lemma,
                  relevant = x$upos %in% c("NOUN", "PROPN", "VERB", "ADJ"),
                  ngram_max = 8, sep = " ")
stats <- subset(stats$keywords, ngram > 1 & freq >= 2)
head(stats)
```

Figure 54: Textrank wordcloud for Surah Yusuf

Figure 54 shows that the keywords combine words into multi-word expressions. Again we see the dominance of the verb *"say"* since Surah Yusuf is a narrated story. It is welcoming to note that "fear Allah" and "do good" are in fact the top moral lessons from this Surah.

> "We relate to you the best of stories through Our revelation of this Quran, though before this you were totally unaware of them." [12:3]

4.5 Dependency parsing

Finally, we address the subject of dependency parsing. Dependency parsing is an NLP technique that provides to each word in a sentence the link to another word in the same sentence, which is called the syntactical head. This link between every two words furthermore has a certain type of relationship giving us further details about it.

The *udpipe* package provides such a dependency parser. With the output of dependency parsing, we can answer questions like

- What is the nominal subject of a text
- What is the object of a verb

[55] https://cran.r-project.org/web/packages/textrank/index.html

- Which word modifies a noun
- What is the link to negative words
- Which words are compound statements
- What are noun phrases, verb phrases in the text

Here we use the dependency parsing output to get the nominal subject and the adjective. When we executed the annotation using *udpipe*, the *dep_rel* field indicates how words are related to one another. A token is related to the parent using *token_id* and *head_token_id*. The *dep_rel* field indicates how words are linked to one another. The type of relations is defined on the Universal Dependencies website.[56] Here we are going to take the words which have a dependency relation *nsubj* indicating the nominal subject and we are adding to that the adjective which is changing the nominal subject.

In this way, we can combine what the Surah is talking about with the adjective or verb it uses when it describes a subject.

```
stats <- merge(x, x,
           by.x = c("doc_id", "paragraph_id", "sentence_id", "head_token_id"),
           by.y = c("doc_id", "paragraph_id", "sentence_id", "token_id"),
           all.x = TRUE, all.y = FALSE,
           suffixes = c("", "_parent"), sort = FALSE)
stats <- subset(stats, dep_rel %in% "nsubj" &
                  upos %in% c("NOUN", "PROPN") &
                  upos_parent %in% c("VERB", "ADJ"))
stats$term <- paste(stats$lemma, stats$lemma_parent, sep = " ")
stats <- udpipe::txt_freq(stats$term)
data.frame("keyword"= stats$keyword,"left" = stats$left,
           "right"= stats$right, "pmi" = stats$pmi )
```

We can visualize the dependency in a wordcloud plot.

Figure 55: Dependency parsing wordcloud for Surah Yusuf

The plot in Figure 55 confirms the comment we made earlier about "say". Another known moral lesson from Surah Yusuf, "patience fitting", now appears. We have shown how to use the *dep_rel* parameter that is part of the annotation output from the *udpipe* package. For visualizing the relationships between the words which were found, we can just use the *ggraph* package. Now we introduce a basic function that selects the relevant columns from the annotation and puts it into a graph as guided by Wijffels et al. (2020) [57]. The code for the function is reproduced as a reference in the Appendix at the end of the chapter.

We can now call the function as follows to plot verse 12:16 in Surah Yusuf. See Figure 56. And a longer verse, verse 12:31. (See Figure 57).

[56] http://universaldependencies.org/u/dep/index.html
[57] http://www.bnosac.be/index.php/blog/93-dependency-parsing-with-udpipe

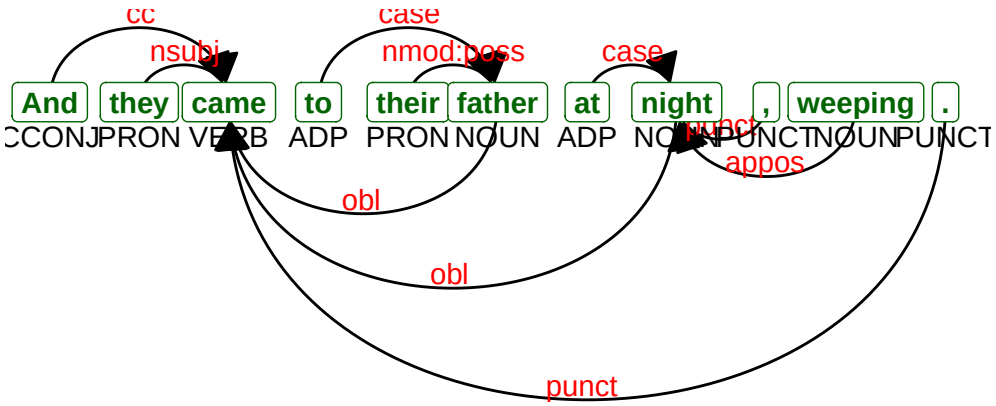


Figure 56: Dependency parsing udpipe output Verse 12:16

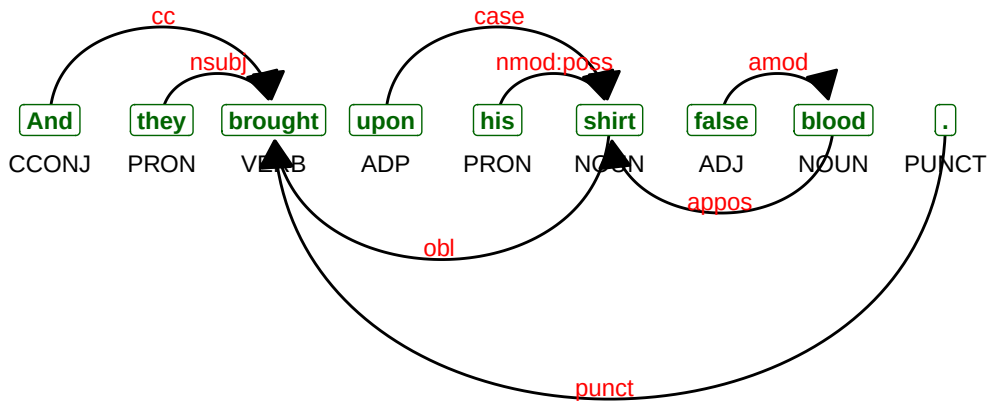


Figure 57: Dependency parsing udpipe output Verse 12:31

With dependency parsing, we can now see, for example, how nouns relate to the adjectives with nominal subject as the type of relationship.

4.5.1 Collocations and co-occurences

Once we have performed the dependency parsing (i.e., all the taggings) for the corpus (or selected subset within the corpus), we can redo all the exercises done in the earlier part of the chapter; namely the collocation measures of "n-grams", "skip-grams", and the co-occurrence measures such as words correlations, and fitting them into a network.

Now that we have classified each word as "NOUN", "PRONOUN", "VERB", etc., we can perform the exercises such as collocations among nouns and pronouns; between verbs, or combing noun and verbs. The permutations are almost limitless. Since our intent is only to introduce the subject and the tools available, we will leave it for further research, in particular for researchers with a background in linguistics.

4.6 Summary

This initial exploration of the *udpipe* package with just one of the 114 Surahs in Saheeh translation of Al-Quran has indeed shown some interesting and unique analysis. The results confirm many familiar lessons for those acquainted with the Quran and Surah Yusuf, in particular.

The study also opens other investigation avenues like

1. Looking into other Surahs of the Quran,
2. Other translations,
3. Other use cases of *udpipe*,
4. Most importantly, analyzing the original Arabic Quran.

The first in the "to-do" list should be easy since the codes can be repeated by just changing the selected Surah to analyze. We encourage our readers to pursue this.

The second can also be easily tested with other English translations of the Quran like what we have shown in Chapter 2 and Chapter 3. Again we leave this to our readers. It can also be repeated with translations of the Quran in other languages that are supported by udpipe.[58]

We will explore the third item in the coming chapters. The final "to-do" item is massive and extremely valuable. We must define a full project scope for it. Applying the NLP tools that we have covered so far, is the easier part, but interpreting the results will be the real challenge. Much of this work has been done by Corpus Quran project[59]; the work of which still requires intensive development and verification of its accuracies and appropriateness.[60]

In the next chapter, we will go deeper into the tools of network graphs in **R**. We have used it quite heavily in this chapter and also in some of the earlier chapters. Our work on Quran Analytics will use these tools frequently. As such, a simple tutorial on these tools using examples from the Quran should be useful.

4.7 Further readings

widyr package in **R**. (Robinson et al., 2020)

udpipe package in **R**. (Wijffels et al., 2020)

[58] https://cran.r-project.org/web/packages/udpipe/vignettes/udpipe-annotation.html

[59] https://corpus.quran.com

[60] Accuracies of the work is still a work in progress, as claimed by the developers, posted on its message board (https://corpus.quran.com/messageboard.jsp)

tm package in **R**. (Feinerer et al., 2020)

textrank package in **R**. (Wijffels and BNOSAC, 2020)

Appendix

Function for Udpipe Dependency Parser output

The codes for this function was produced by bnosac (https://www.bnosac.be).

```
plot_annotation <- function(x, size = 3){
  stopifnot(is.data.frame(x) &
            all(c("sentence_id", "token_id", "head_token_id",
                  "dep_rel", "token", "lemma", "upos",
                  "xpos", "feats") %in% colnames(x)))
  x <- x[!is.na(x$head_token_id), ]
  x <- x[x$sentence_id %in% min(x$sentence_id), ]
  edges <- x[x$head_token_id != 0,
             c("token_id", "head_token_id", "dep_rel")]
  edges$label <- edges$dep_rel
  g <- graph_from_data_frame(
              edges,
              vertices = x[, c("token_id", "token",
                               "lemma", "upos", "xpos", "feats")],
              directed = TRUE)
  ggraph(g, layout = "linear") +
      geom_edge_arc(
        ggplot2::aes(label = dep_rel, vjust = -0.20),
        arrow = grid::arrow(length = unit(4, 'mm'),
                            ends = "last",
                            type = "closed"),
        end_cap = ggraph::label_rect("wordswordswords"),
        label_colour = "red", check_overlap = TRUE, label_size = size) +
      geom_node_label(ggplot2::aes(label = token),
                      col = "darkgreen",
                      size = size, fontface = "bold") +
      geom_node_text(ggplot2::aes(label = upos), nudge_y = -0.35, size = size) +
      theme_graph(base_family = "Arial Narrow") +
      labs(title = "udpipe output",
         subtitle = "tokenisation, parts of speech tagging & dependency relations")
}
```

5 Graph Representations of Word Cooccurences

Word collocations and co-occurrences as explained in Chapter 4 reveal certain dimensions of relations between words. The process allows us to capture and analyze the appearances of words together. This allows linguists to understand what are the meanings of the existence of these words together, from a general linguistic point of view as well as from grammatical perspectives. This is the simplest and most basic method of analyzing word relations (i.e., through collocations and co-occurrences).

Stylistically expressive elements in a text can be identified at the word-level (lexical), in the way sentences are structured (syntactic), and by analyzing the attributes of the core meaning that is conveyed (semantic) (DiMarco and Hirst, 1988). An example of lexical elements is the choice of words between synonyms, e.g. “residence” versus “home”. An example of syntactic elements is sentence compounding, e.g. “We have never been to Asia, nor have we visited Africa”, where a coordinating conjunction is required for two independent clauses to relate. An example of semantic elements in style and expression, is “John attended Oxford, which is the best university in the UK” versus “Oxford, the best university in the UK attended by John”. In the first sentence, the emphasis is on John, while the second emphasizes Oxford.

At the fundamental level, all three elements involve the “positioning” of words in a sentence and how they fit into the style and purpose of the texts in question. Comprehensive linguistic studies require a deep look into the three elements within the texts of the whole corpus to determine differences in styles, methods, and expositions of the authors.

In the case of Al-Quran, we have the original Arabic text, which is fixed and unchangeable. We have various translations of the Quran into other languages, of which the sample English translations of Saheeh and Yusuf Ali are of particular interest in this book. Since a comprehensive analysis requires knowledge and expertise of linguistics, of which neither of us is an expert, our focus is on exhibiting and highlighting the possible issues

in general without adhering to any linguistic or language rules. This is what we term as a "non-parametric" approach that we alluded to in the introduction of the book.

This chapter is divided into three main parts. In the first part, we deal with statistical properties of the word positions and perform comparisons between the Arabic text (Quran Arabic) and the English translations (Saheeh and Yusuf Ali). This is the straightforward way of dealing with the statistical properties of word relations. In the second part, we will take a different approach in dealing with the same issue, by converting the data into a network graph structure, using Surah Yusuf as our sample. Finally, in the third part, we will provide a short tutorial on *igraph* and *ggraph* packages in **R** and use Surah Yusuf as a working example for various network analyses which are useful for the coming chapters of the book.

5.1 Statistical analysis of word positions

Grammatical and syntactical positions or POS tags within a corpus show the usage of grammar styles in the writing of the author. Since the original source of the text is Al-Quran Arabic, upon which all translations are based, one way to observe the differences in the styles is by comparing the statistical properties of these POS tags between the corpora. The properties of these tags may reveal differences in style and approach of the texts, as explained earlier.

We will do the analysis based on Saheeh, Yusuf Ali and we will use the Quran Arabic as benchmark comparisons whenever required.[61] First, we create a *data.frame* for the entire Saheeh corpus *udpipe* annotation for the English language model (named *QSI_udp* for Saheeh and *QYA_udp* for Yusuf Ali). Then we also create *udpipe* annotation for the Arabic language model (named *QAR_udp* for Quran Arabic text without diacritics (from *quran_ar_min* of *quRan* package))[62].

5.1.1 Comparison between Saheeh and Yusuf Ali

One simple and basic analysis is to observe the percentages of UPOS across segments of texts, such as chapters, and in our case Surahs, within a corpus (Saheeh or Yusuf Ali) and compare with another corpus (Saheeh vs Yusuf Ali).

```
early_surah_no = c("2","3","4","5","6")
mid_surah_no = c("56","57","58","59","60")
last_surah_no = c("107","108","109","110","111","112","113","114")
pos_plotter = function(df_udp, surah_no_input,title_label){
  df_udp %>% filter(surah_no %in% surah_no_input) %>%
                group_by(surah_no) %>% count(upos) %>%
                mutate(prop=n/sum(n)) %>%
                ggplot(aes(x = as.factor(upos), y=prop, color = surah_no)) +
                  geom_point() + scale_color_brewer(palette = "Dark2") +
                  theme(axis.text.x = element_text(angle = 90),
                        legend.position = "top") +
                  ylim(0,0.35) +
                  labs(title = title_label,
                     x = "UPOS", y = "percentage")
}
```

First, let us plot the POS percentages for the entire text.

From Figure 58, we can see that both Saheeh and Yusuf Ali prominently use nouns, pronouns, verbs, and adpositions almost similarly (based on the observation for the texts, the full sample size). However, in the case of punctuation, Yusuf Ali tends to be higher than Saheeh. As indicated by linguists, usage of more punctuation indicates the possible difference between the literary English of Yusuf Ali compared to Saheeh (possibly due to the American English of Saheeh).

[61] A full-scale work for the Arabic language is beyond the scope of this book and will be left as future research.

[62] We have not analyzed the performance of the Udpipe model for the Arabic language, hence we cannot ensure its accuracies.

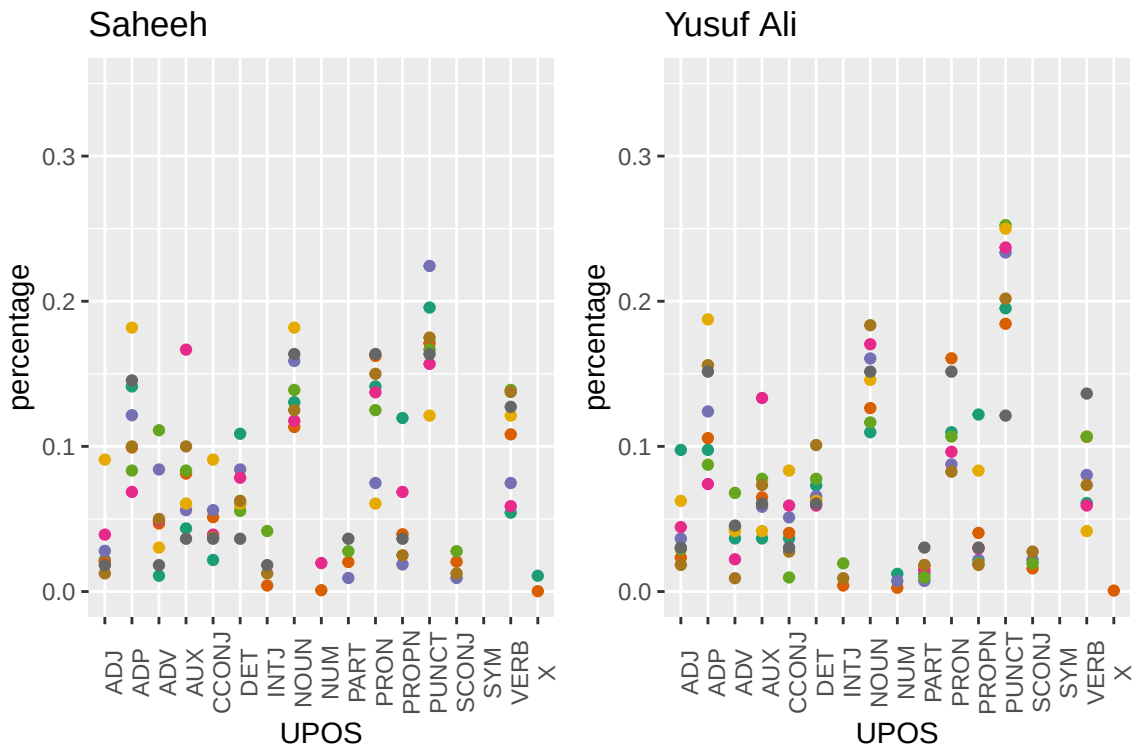


Figure 58: Percentage of POS categories in Saheeh and Yusuf Ali for the entire text

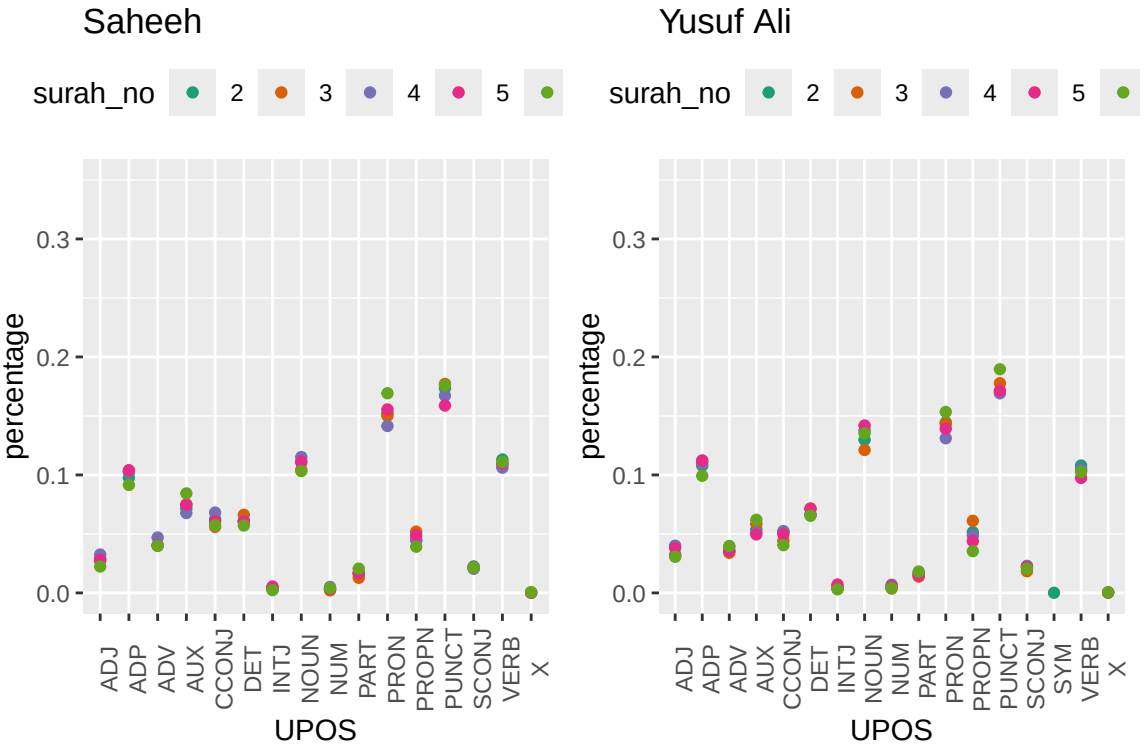


Figure 59: Percentage of POS categories in Saheeh and Yusuf Ali for early Surahs

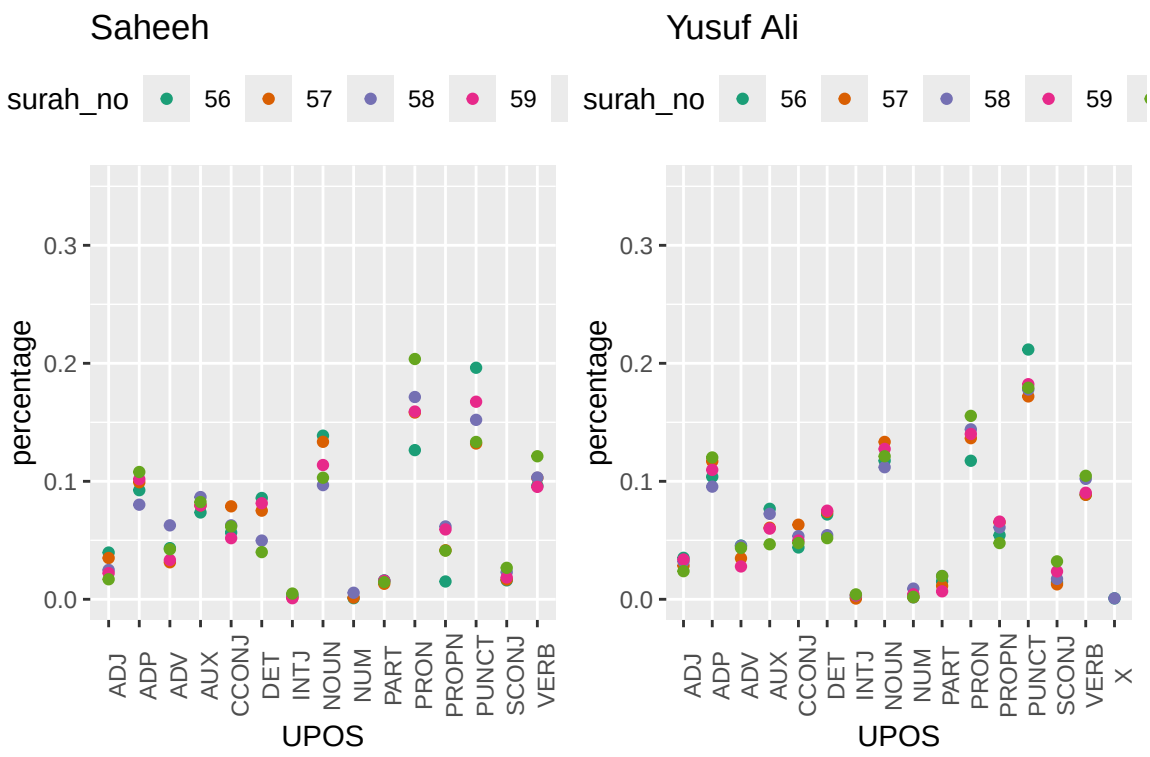


Figure 60: Percentage of POS categories in Saheeh and Yusuf Ali for middle Surahs

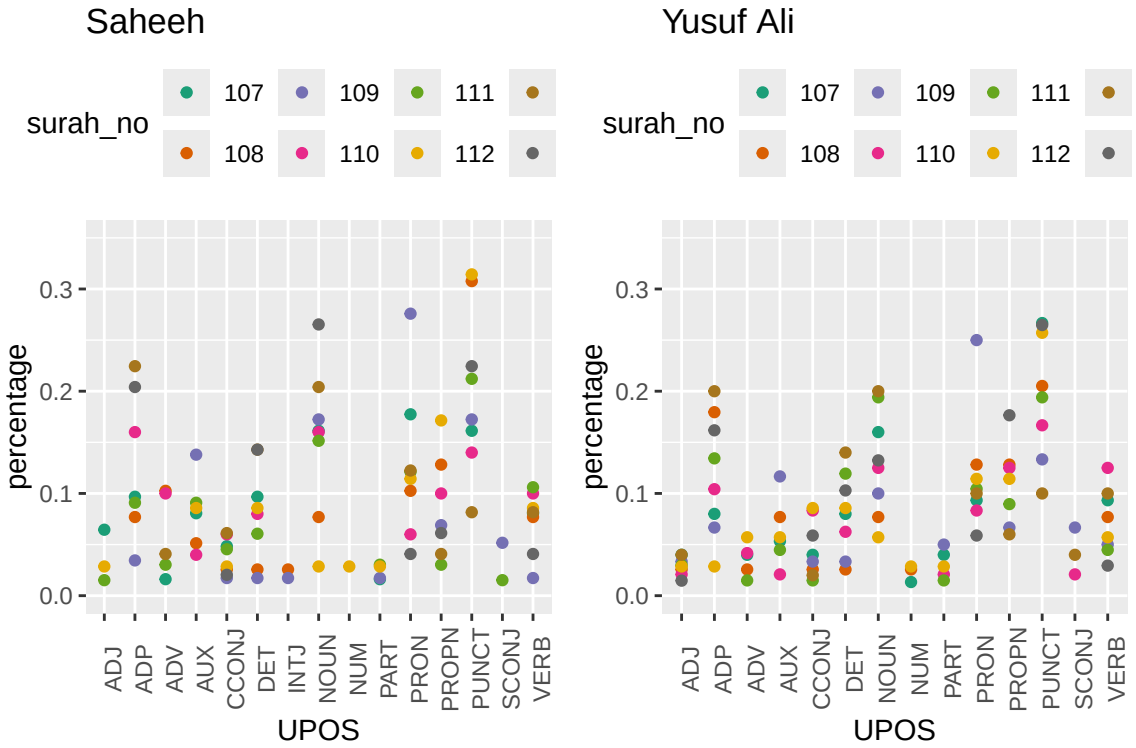


Figure 61: Percentage of POS in Saheeh and Yusuf Ali for last Surahs

The plots resulting from the codes are in Figure 59, Figure 60, and Figure 61. They refer to the long Surahs, medium Surahs, and short Surahs (the last few). We will make some observations based on just visualizing the plots.

In general, we can conclude that, based on the statistics of POS tags, Saheeh and Yusuf Ali differ to some degree from a lexical perspective. A clear trend emerges where, as the Surahs become shorter, the compositions of *upos* tags vary with higher variability. This re-confirms our earlier observations from Figure 14 in Chapter 2, which shows that the lexical variety increases as the Surah becomes shorter.

Furthermore, the lexical differences between the translations may extend into the semantics as well the pragmatics, which indicate differences in meaning or interpretation. Again, this is a non-trivial issue, since over the years many attempts to produce reliable interpretations of Al-Quran have been attempted by many scholars. They need to be re-looked upon since it may seriously affect the teaching of Al-Quran using interpretations based upon non-Quranic native languages.

5.1.2 Comparison against the Arabic text

We will compare the POS taggings with the Arabic text, for the purpose of benchmarking. It is not our intention to do a comprehensive analysis of the texts, as it is beyond the scope of the current book. We have planned it for our future research. However, we believe that if we want to compare the works of translations, a benchmark is required, and there is no better choice than the original Arabic text itself.

First we need to load the Arabic POS tagging data[63] into the environment and then we can make the comparisons.

We first tabulate the data in numbers for a quick overview.

	Arabic	Saheeh	Yusuf Ali
PUNCTUATION	18,157	33,960	39,315
NOUN	40,093	21,754	26,251
PRONOUN	29,940	29,986	29,612
VERB	31,337	20,723	21,305
X	34,701	55	66
ADPOSITION	30,721	18,614	21,722
Total tokens	67,161	113,370	117,415
Unique tokens	16,208	5,739	7,484

Yusuf Ali is consistently higher than Saheeh in most categories: punctuation, nouns, pronouns, verbs, and adpositions. If we benchmark the English translations against the Arabic, we can see that both use fewer nouns and pronouns combined. Even the verbs are greater in Arabic compared to the English translations.

Clearly, the English translation (represented by Saheeh and Yusuf Ali) is starkly different compared to the Arabic when it comes to the lexical composition (which is probably obvious, due to the difference between English and Arabic). However, is the difference due to the efforts of translating compact Arabic texts to English, or is it due to the compactness of meaning? The first difference is purely a question of lexical styles, whilst the second one is a question of semantics and pragmatics of the language and texts. The answer to this question is definitely non-trivial and requires deeper research.

Based on the analysis here, we can provide some suggestions, which are evident (visually) through the plots of the statistics of the POS tags as presented in Figure 62. For this purpose, we made some comparisons between Saheeh and Arabic (we left out Yusuf Ali for brevity) and included comparisons for the various Surahs' groupings in Figure 63, Figure 64, and Figure 65 for the readers to make some sense out of it.

[63] The data is obtained from running *udpipe* Arabic padt model on the *quran_ar_min* dataset from the *quRan* package.

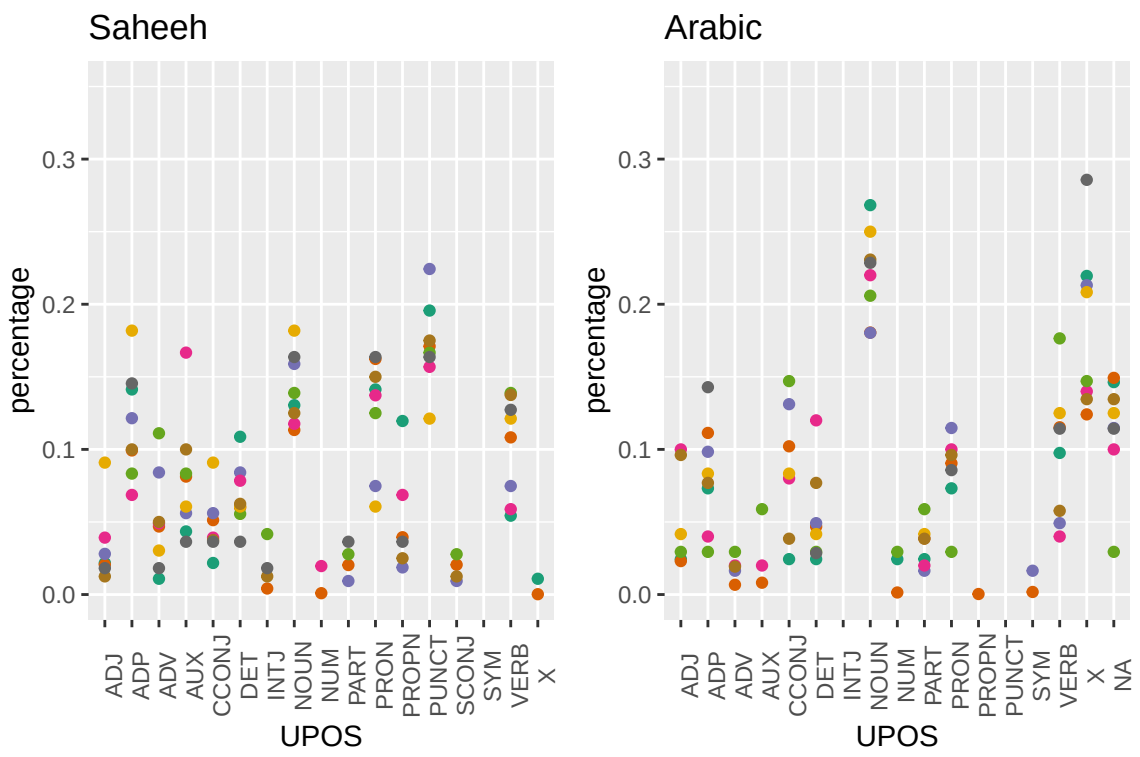


Figure 62: Percentage of POS categories in Saheeh and Arabic for the entire text

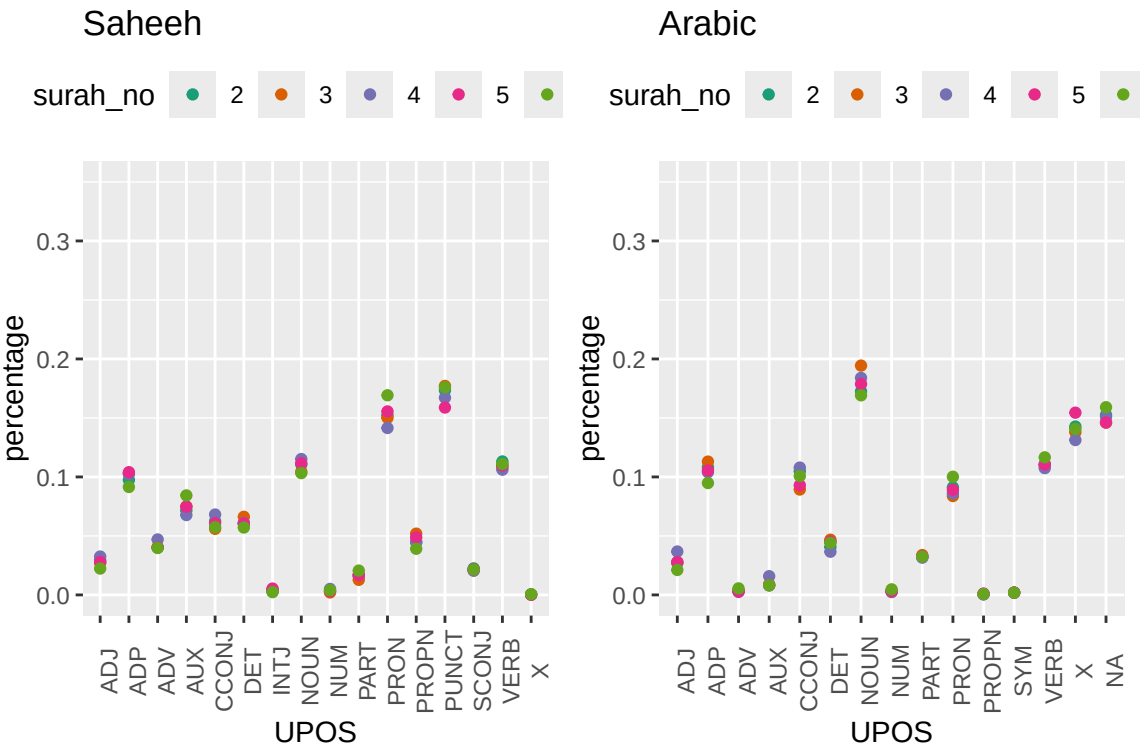


Figure 63: Percentage of POS categories in Saheeh and Arabic for the early Surahs

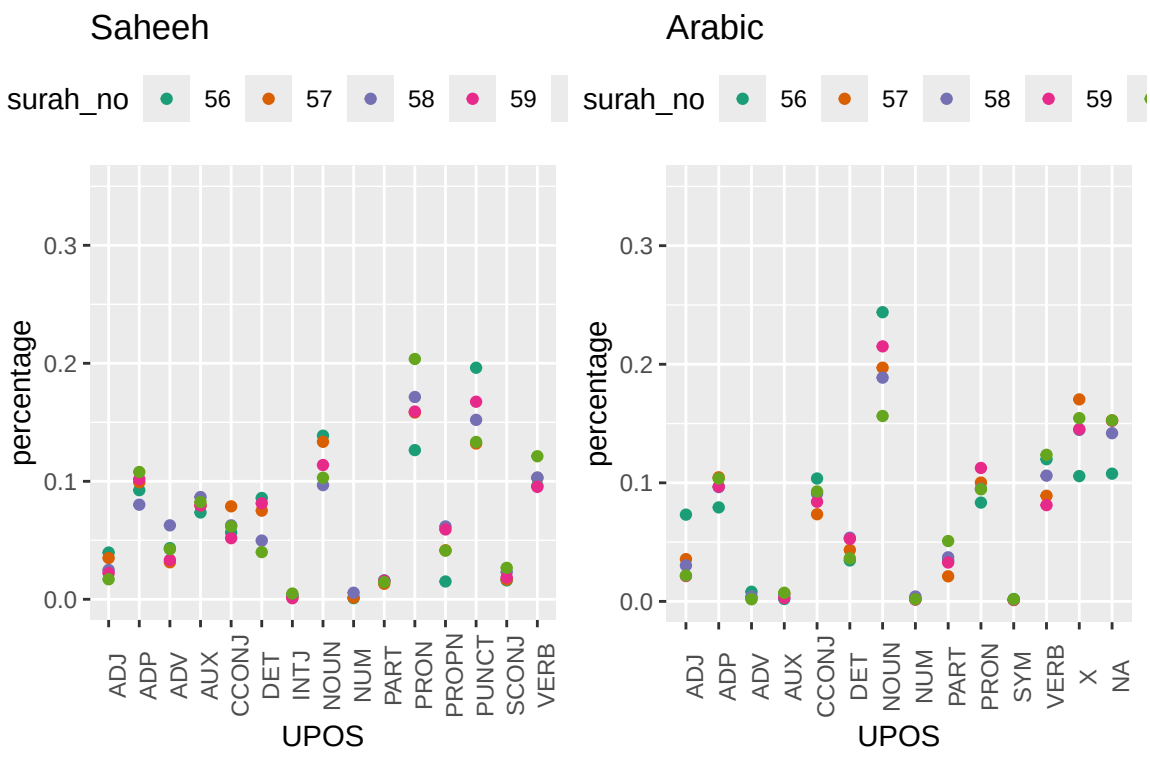


Figure 64: Percentage of POS categories in Saheeh and Arabic for the middle Surahs

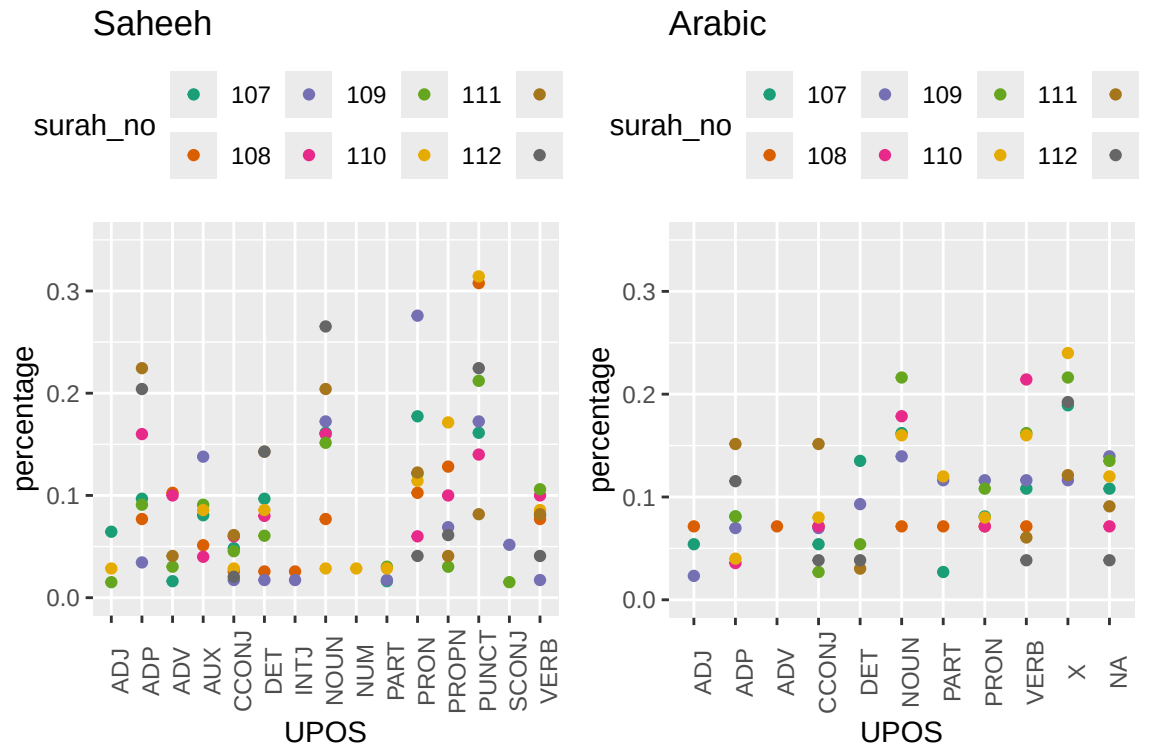


Figure 65: Percentage of POS categories in Saheeh and Arabic for the last Surahs

Finally, we leave with the following question: what do all of these mean when translating the Quranic Arabic to the English language (and other languages for that matter)? At least we may say so from the semantic styles point of view, as indicated by many Islamic scholars that Al-Quran has a special language (Saeh, 2015). But, is it also true for lexical and syntactic styles? Furthermore, does the semantic richness of the Arabic used in Al-Quran, together with the lexical and syntactic richness combined, result in texts which produce higher-level meanings, implying that to learn Al-Quran, one must rely only on the original Arabic text?

General comparisons as provided in this section open up more questions than answers. This is exactly the purpose of our book, to ask questions from the exploratory findings, from which we suggest other research topics.

5.2 Focus on Surah Yusuf

Now we will explore in more detail the observations made in the previous section, applied to a specific Surah, Surah Yusuf. The choice of Surah Yusuf is made consciously with the knowledge that the Surah contains one major story, namely that of Prophet Yusuf (Joseph).

We will start by plotting the POS tags statistics for the Surah Yusuf from Saheeh and Yusuf Ali.

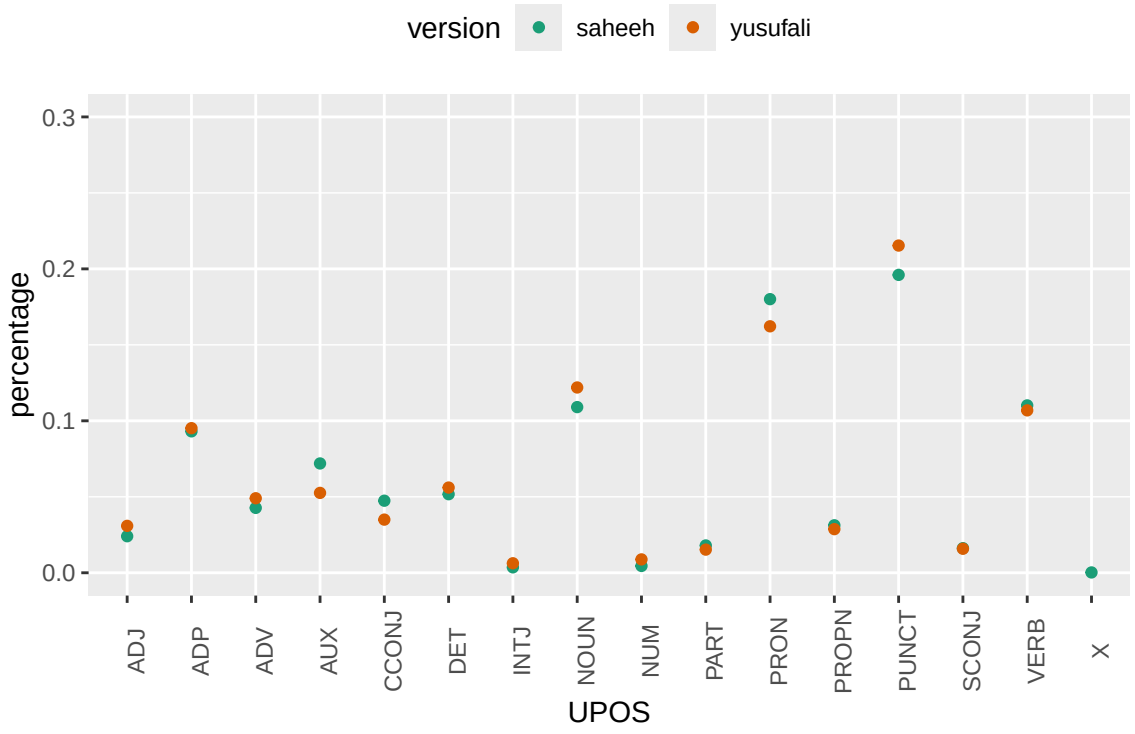


Figure 66: Percentage of POS categories in Surah Yusuf

As evident from Figure 66, the differences in lexical styles between Saheeh and Yusuf Ali are not that obvious, except for the fact that Yusuf Ali used more punctuations, which is the same observation made for the entire Quran (as in Figure 58).

Now we will start to compute the co-occurrences in Surah Yusuf for the Saheeh and Yusuf Ali.

```
sy_si = QSI_udp %>% filter(surah_no %in% "12")
sy_ya = QYA_udp %>% filter(surah_no %in% "12")
QSI_cooc <- cooccurrence(x = subset(sy_si, upos %in% c("NOUN", "PROPN", "VERB")),
                         term = "lemma",
```

```
                      group = c("ayah_no", "paragraph_id", "sentence_id"))
QYA_cooc <- cooccurrence(x = subset(sy_ya, upos %in% c("NOUN", "PROPN", "VERB")),
                      term = "lemma",
                      group = c("ayah_no", "paragraph_id", "sentence_id"))
```

We then create a graph from the *data.frame* as an *igraph* graph object, which we can visualize as a graph object using *ggraph*.

```
QSI_cooc_g <- graph_from_data_frame(head(QSI_cooc,100))
QYA_cooc_g <- graph_from_data_frame(head(QYA_cooc,100))
gg_plotter1 = function(grf_plot){
  ggraph(grf_plot, layout = "fr") +
  geom_edge_link(aes(width = cooc, edge_alpha = cooc),
                 edge_colour = "deeppink") +
  geom_node_point(aes(size = igraph::degree(grf_plot)), shape = 1,
                  color = "black") +
  geom_node_text(aes(label = name), col = "darkblue", size = 3)
}
```

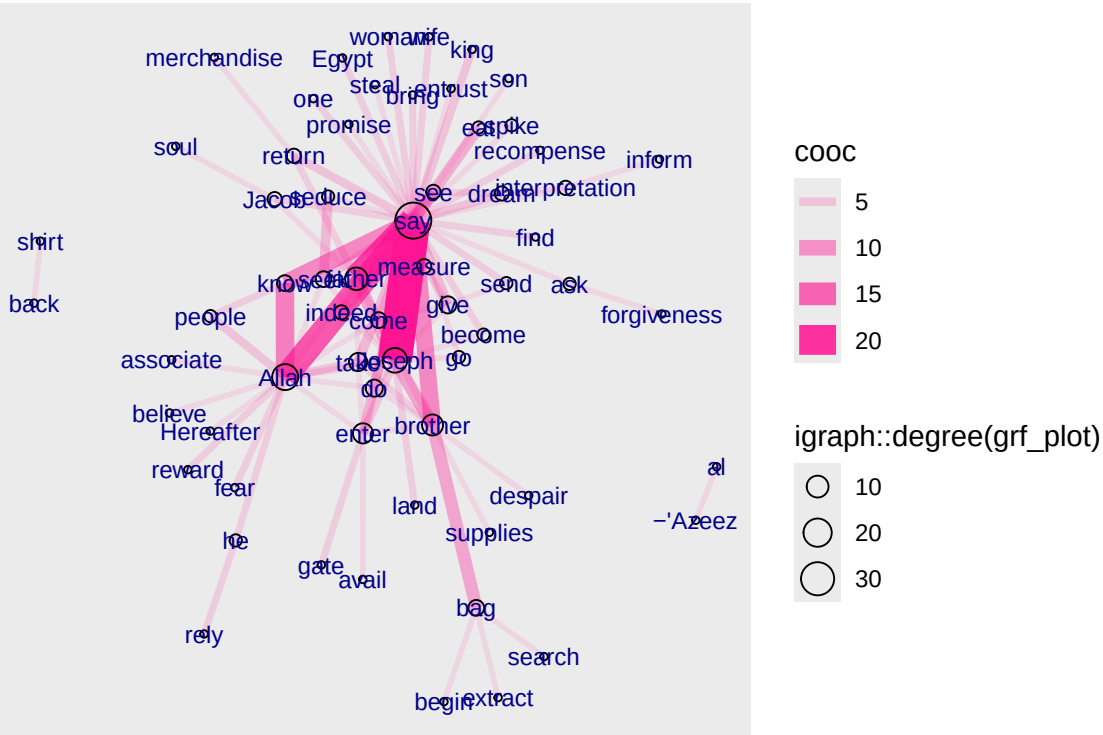


Figure 67: Co-occurrence network for top relations in Surah Yusuf for Saheeh

The plots in Figure 67 and Figure 68 are networks of words that co-occur frequently within Surah Yusuf. The nodes (the words), which are sized to their frequency of appearances, are linked by the edges, which are sized to the number of times the co-occurrence occurs. Clearly in the Saheeh and Yusuf Ali version, "say", "Allah" and "Joseph" are higher by both counts. This is similar to the bigrams we did earlier with one major difference: we tag the words in accordance to its dependency parser and no stopwords are removed from the texts.[64]

[64]This is an important point to be made since we alluded to earlier that removal of stopwords while being practiced frequently in NLP analysis, is non-trivial; and should only be used as a last resort rather than by default.

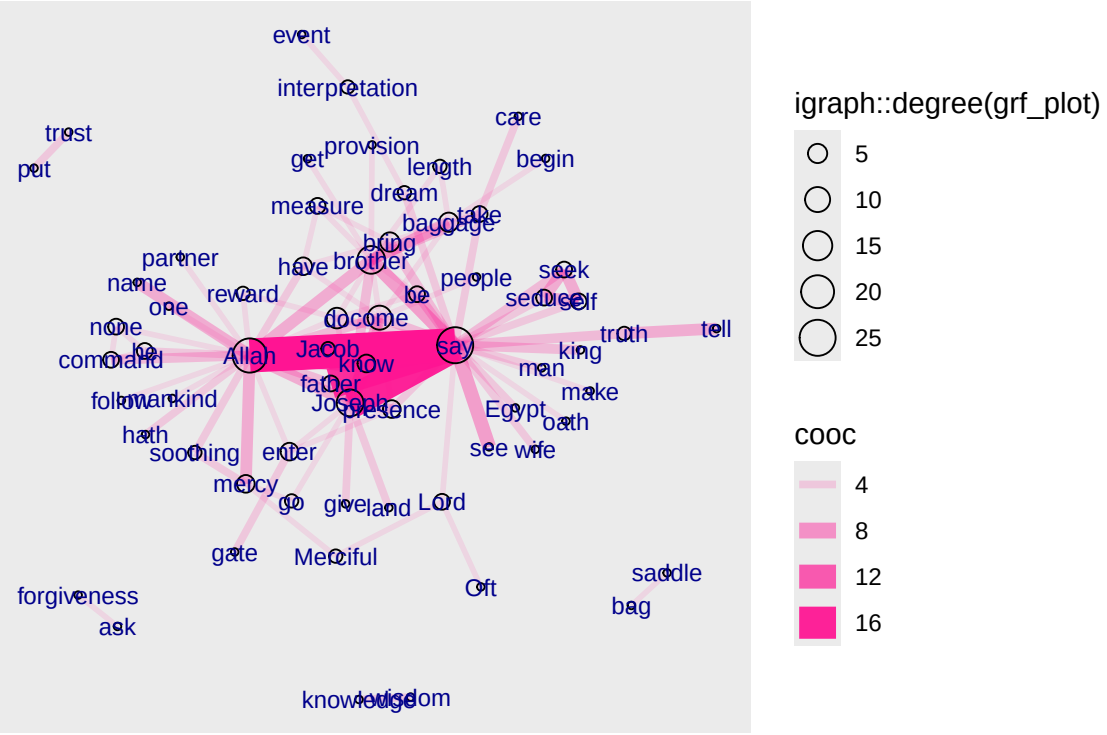


Figure 68: Co-occurrence network for top relations in Surah Yusuf for Yusuf Ali

5.2.1 Arc method of visualization

Graphs in the sense of graph theory, represent a powerful mathematical concept as well as a beautiful way of visualizing data. Here we present a few different ways of presenting the same data but using different layouts.

```
ggraph(QSI_cooc_g, layout = 'linear') +
    geom_edge_arc(color = "gold3", width=0.5) +
    geom_node_point(size=2, color="black") +
    geom_node_text(aes(label = name), repel=TRUE, size = 4) +
    theme_void()
```

Figure 69 and Figure 70, are "arc" layouts, which organize the links in a "conversational" manner. We can observe that Saheeh's groupings of relations differ from Yusuf Ali's (from the organization of the arcs). The question is which one is nearer to the original texts (as shown in Figure 71). We let the readers make their own judgment!

What we want to show is there are many tools besides pure statistical analysis to see the lexical styles of texts, and possibly the semantic styles, by just observations through visualizations.

5.2.2 Circular method of visualization

```
ggraph(QSI_cooc_g, layout = 'linear', circular = TRUE) +
     geom_edge_arc(color = "gold3", width=0.5) +
     geom_node_point(aes(size = degree(QSI_cooc_g)),
                     alpha = igraph::degree(QSI_cooc_g),
```

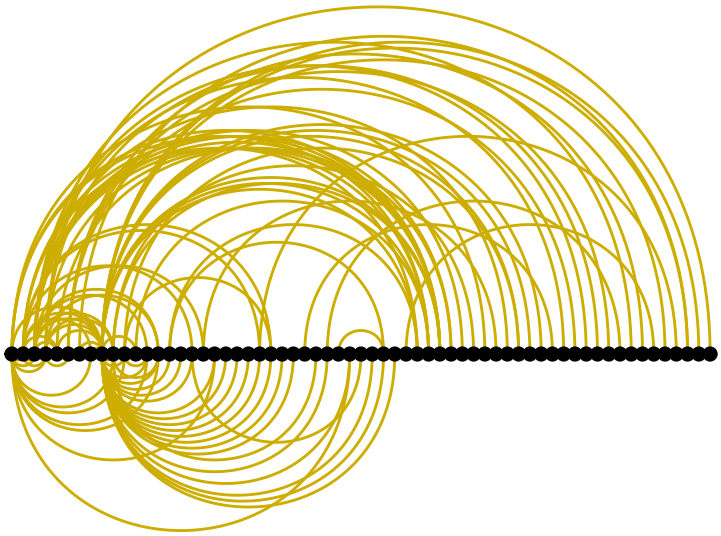


Figure 69: Arc view of co-occurrence network for top relations in Surah Yusuf for Saheeh

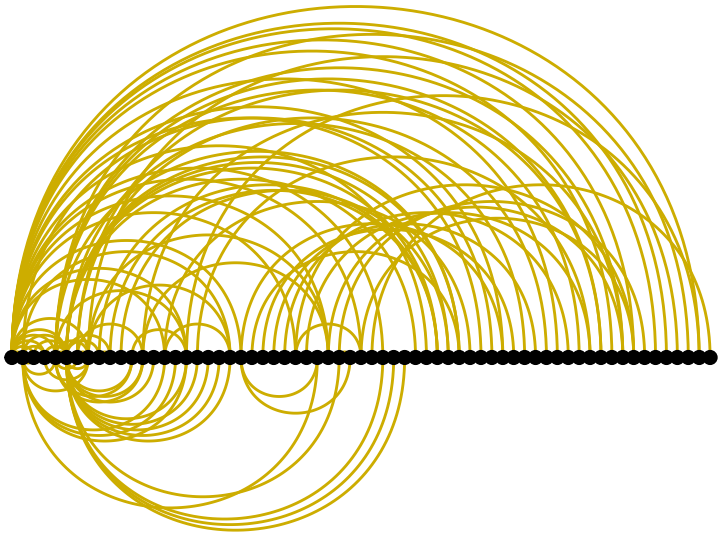


Figure 70: Arc view of co-occurrence network for top relations in Surah Yusuf for Yusuf Ali

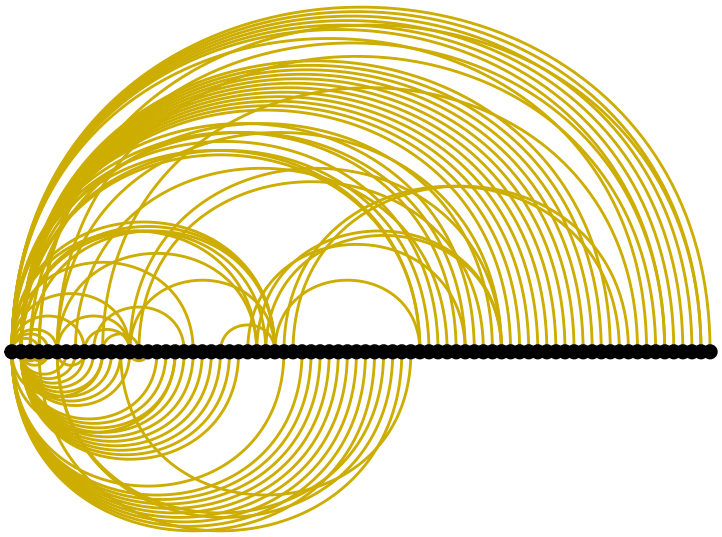


Figure 71: Arc view of co-occurrence network for top relations in Surah Yusuf for Arabic

```
                    colour = "#a83268") +
  geom_node_text(aes(label = name), size = 3, repel=TRUE) +
  theme_void()
```

Figure 72 and Figure 73, are circular layouts that provide another perspective of viewing, like a round-table discussion. Do the two "round tables" (Saheeh versus Yusuf Ali) look the same? We can say that they are close, but not exactly the same. This is an example of the subtle differences in the "semantic meaning" (and may also be pragmatic meaning) of the texts.

For benchmarking, let us just compare both to the Arabic version in Figure 74 and we let the readers make their own judgment.

5.2.3 Grouping of co-occurences

Next, we can ask, do these words which co-occur highly have their groupings? We know offhand that there are few nodes (words) that will have links to many other words, but we want to see whether all these "smaller" nodes (words of lesser prominence) are grouped in a certain manner.

This is easily visualized through graph clustering algorithms. We will show two different methods of "graph clustering", one is based on the *fastgreedy* algorithm.

```
fc1 <- fastgreedy.community(simplify(as.undirected(QSI_cooc_g)))
set.seed(1234)
plot(QSI_cooc_g, vertex.color=vertex_attr(QSI_cooc_g)$cor,
      vertex.size = degree(QSI_cooc_g),
      vertex.label=NA,
      edge.width = NA,
      edge.color = NA,
      layout = layout_with_kk,
      mark.groups = fc1,
      mark.border=NA)
```

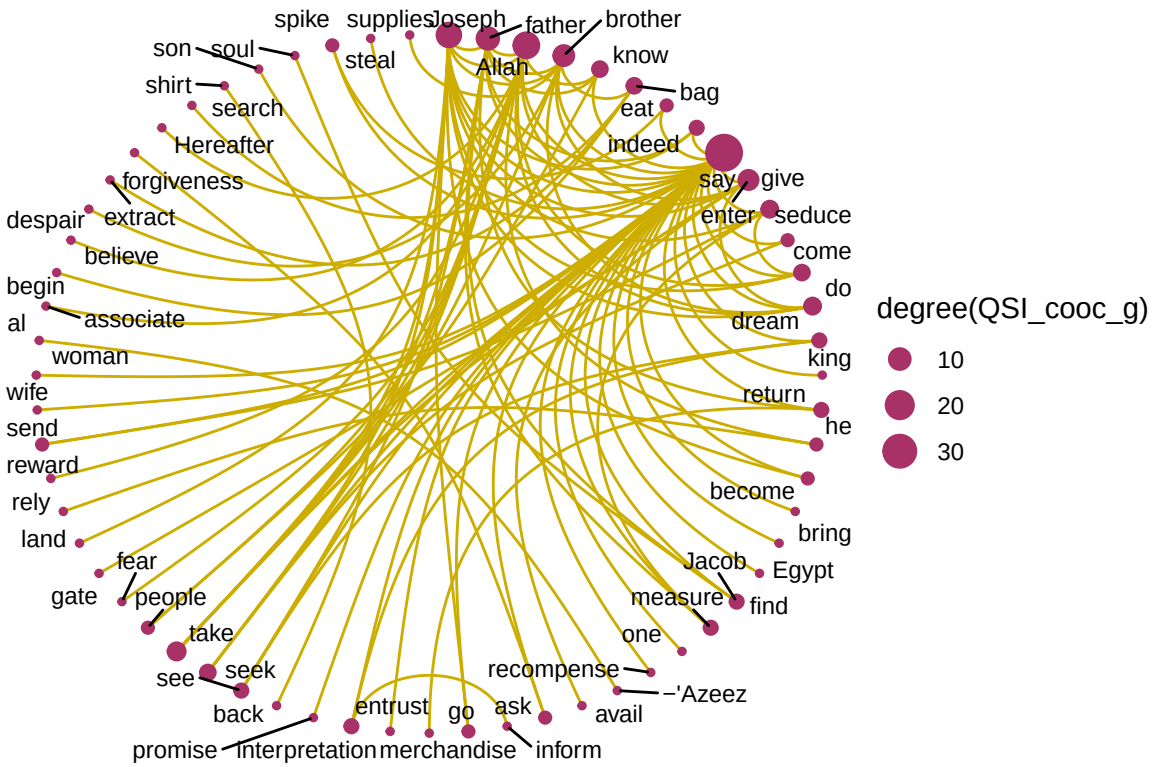


Figure 72: Circular view of co-occurrence network for top relations in Surah Yusuf for Saheeh

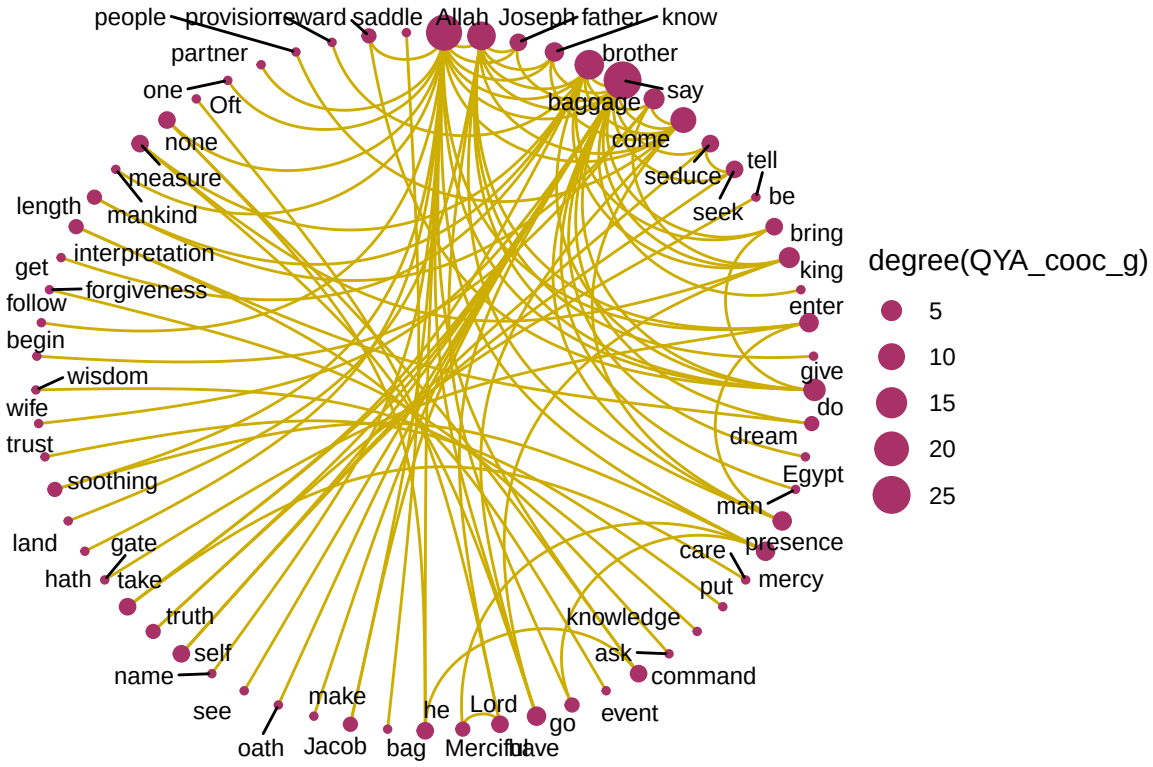


Figure 73: Circular view of co-occurrence network for top relations in Surah Yusuf for Yusuf Ali

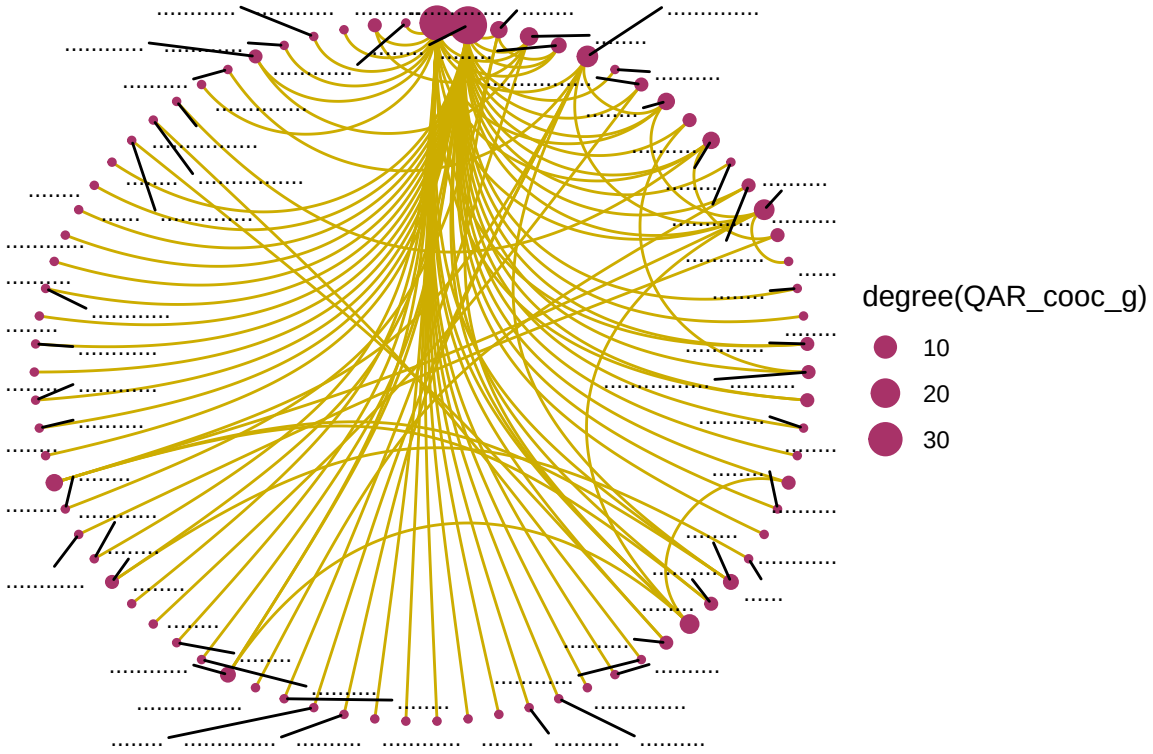


Figure 74: Circular view of co-occurrence network for top relations in Surah Yusuf for Arabic texts

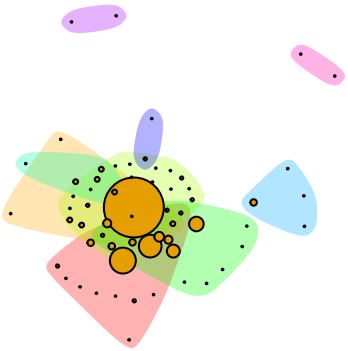


Figure 75: Groupings by fastgreedy in Surah Yusuf for Saheeh

```
fc2 <- fastgreedy.community(simplify(as.undirected(QYA_cooc_g)))
set.seed(2345)
plot(QYA_cooc_g, vertex.color=vertex_attr(QYA_cooc_g)$cor,
      vertex.size = degree(QYA_cooc_g),
      vertex.label=NA,
      edge.width = NA,
      edge.color = NA,
      layout = layout_with_kk,
      mark.groups = fc2,
      mark.border=NA)
```

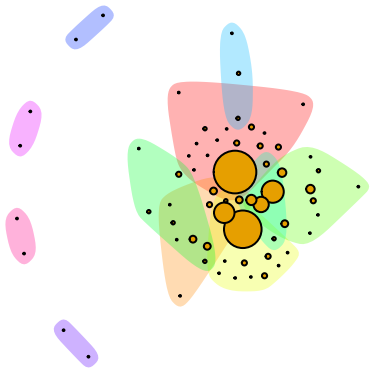


Figure 76: Groupings by fastgreedy in Surah Yusuf for Yusuf Ali

Figure 75 and Figure 76 show the groupings by the colors. To interpret these visuals, we use the analogy of people in a large ballroom having conversations or hearing conversations. Are there any groupings between the people (denoted by the color schemes)? The larger nodes represent people who speak more than the smaller nodes, and a set of colors represent the sub-groupings of people. Based on this analogy, we can see that Saheeh differs from Yusuf Ali to some degree. What this implies is that the differences between Saheeh and Yusuf Ali are beyond lexical, syntactic, and semantic, rather it goes towards pragmatic (i.e. dynamics of conversations) as well.

There are so many ways to show these dynamics using graph theory mathematics, but for now, it suffices to leave the readers with visual impressions. For this reason, in the next section, we present a short tutorial on graphs in **R**. We want the readers to use some of the methods of graphing and learn to see many other dimensions of the relations using graph algorithms.

5.3 A short tutorial on graphs in R

In this section, we will detour from our main discussion to provide a brief tutorial for graph packages in **R**. We will use the data from Surah Yusuf that we worked on earlier as a sample. This serves both purposes of showing how we can use **R** for analysis using network graphs, and at the same time demonstrate various possibilities of analysis using graph packages in **R**.

We will start with the data. As explained in the previous chapter, word co-occurrences allow us to see how words are used either in the same sentence or next to each other. The *udpipe* package makes creating co-occurrence graphs using the relevant POS tags easy. We look at how many times nouns, proper nouns, adjectives, verbs, adverbs, and numbers are used in the same verse.

```
cooccur <- cooccurrence(x = subset(x,upos %in% c("NOUN", "PROPN", "VERB",
                                                 "ADJ", "ADV", "NUM")),
                        term = "lemma",
```

```
                      group = c("doc_id", "paragraph_id", "sentence_id"))
head(cooccur,10)
```

The result can be easily visualized using the *igraph* and *ggraph* R packages as we have seen earlier.

```
library(igraph)
library(ggraph)
library(ggplot2)
wordnetwork <- head(cooccur, 100)
wordnetwork <- graph_from_data_frame(wordnetwork)
ggraph(wordnetwork, layout = "fr") +
  geom_edge_link(aes(width = cooc, edge_alpha = cooc), edge_colour = "#ed9de9") +
  geom_node_point(aes(size = igraph::degree(wordnetwork)), shape = 1,
                  color = "black") +
  geom_node_text(aes(label = name), col = "darkblue", size = 3) +
  labs(title = "Co-occurrences within sentence",
       subtitle = "Top 100 Nouns, Names, Adjectives, Verbs, Adverbs",
       caption = "Surah Yusuf (Sahih International)")
```

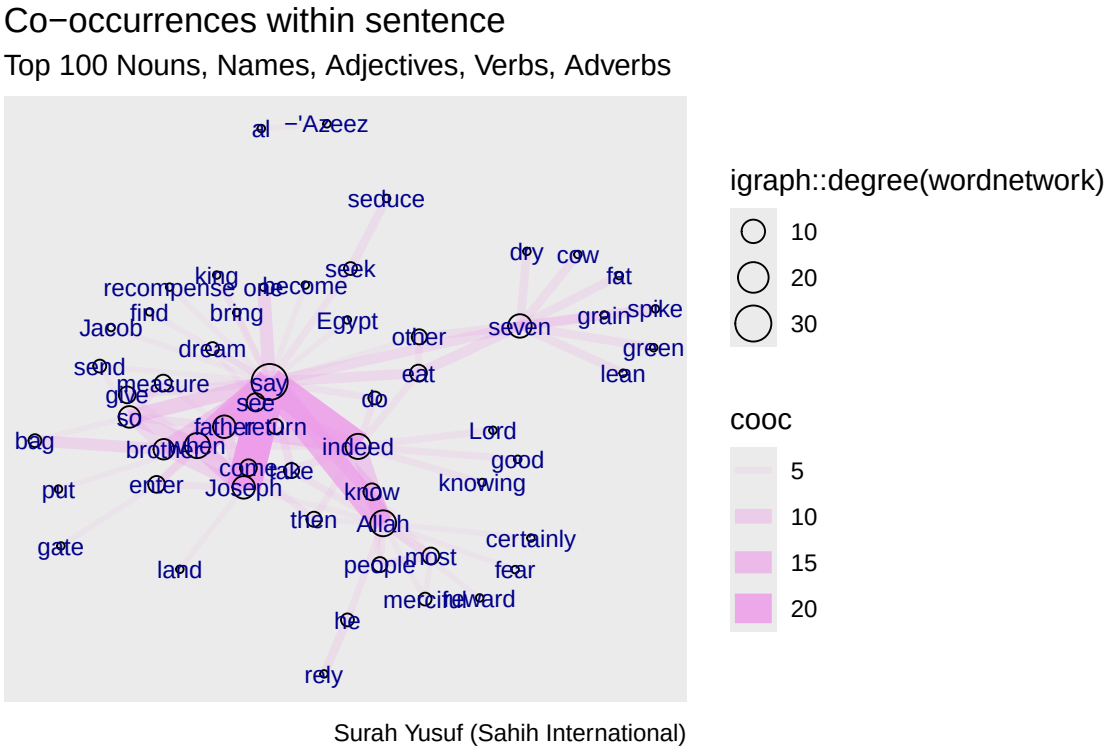


Figure 77: Co-occurence within sentence

The network graph shows the main words used in the Surah as nodes. The nouns and proper nouns like Joseph, his father, his brothers, the king, and the wife of the minister (al-'Azeez) make up the main characters in this revealed story. The strength or influence of each word (node) is visualized by the size of the circle representing the words (nodes) and the thickness of the links with other nodes. The verb "say" is a node that dominates since Surah Yusuf is a narrated story. It is interesting to see the strong link and occurrence of "know" with "Allah".

After the simple introduction and sample examples, we come back to the graph model of how many times

nouns, proper nouns, adjectives, verbs, adverbs, and numbers are used in the same verse in Surah Yusuf (111 verses).

5.3.1 Graph creation

Data, format, size The example data used here is not big, many of the ideas behind the visualizations we will generate apply to medium and large-scale networks.

- When drawing very big networks we focus on identifying and visualizing communities or subsets of nodes.
- Visualizing larger networks as giant hairballs is less helpful than providing plots that show key characteristics of the graph.

Create graph

- Read nodes (term1 and term2) and edges
- Convert raw data to an *igraph* network object.
- Use the graph_from_data_frame() function, which takes one data frame (in our case)
  - Its first two columns are the IDs of the source and the target node for each edge.
  - The following columns are edge attributes (cooc, or the number of co-occurrences).
- Nodes start with a column of node IDs.
  - Any following columns are interpreted as node attributes.

```
head(cooccur, 10)
wordnetwork <- head(cooccur, 50)
gm <- graph_from_data_frame(wordnetwork)
```

We purposely limited the graph to 50 co-occurrences (links/edges) to make the plots easy to visualize. R is case sensitive. The choice of Weight/weight or Type/type makes a great difference.

- The description of an *igraph* object starts with four letters:
  - D or U, for a directed or undirected graph
  - N for a named graph (where nodes have a name attribute)
  - W for a weighted graph (where edges have a weight attribute)
  - B for a bipartite (two-mode) graph (where nodes have a type attribute)
- The two numbers that follow **(35 50)** refer to the number of nodes and edges in the graph. The description also lists node and edge attributes, for example:
  - (g/c) - graph-level character attribute
  - (v/c) - vertex-level character attribute
  - (e/n) - edge-level numeric attribute

Some simple commands

- We have easy access to nodes, edges, and their attributes with:

```
# Sample of Edges of the network
E(gm)[1:3]
E(gm)[4:6]
# Sample of Vertices of the network
V(gm)[1:4]
V(gm)[5:8]
# Sample of Edge attribute
E(gm)$cooc[1:15]
# Sample of Vertex attribute
V(gm)$name[1:4]
V(gm)$name[5:8]
```

Find nodes and edges by attribute:

- We can select nodes and edges based on their attributes.

```
V(gm)[name=="Allah"]
E(gm)[cooc > 10][1:4]
E(gm)[cooc > 10][5:8]
# You can also examine the network matrix directly:
gm[1:5,1:5]
```

5.3.2 Graph plots

We have created a scaled down network of only 50 edges from the Surah Yusuf word co-occurrence network. We refer to our tutorial word network as **gm**. Let us make a first simple plot and then improve it in steps. We adapted these from https://kateto.net/network-visualization.[65]

Using igraph plots

- There are sufficient plotting functions in *igraph* to begin with.

```
plot(gm)
```

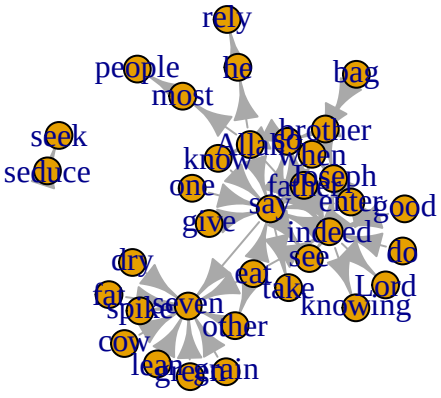


Figure 78: First plot of tutorial word network

Plot output is in Figure 78.

Plotting parameters

[65] Ognyanova, K. (2019) Network visualization with R. Retrieved from www.kateto.net/network-visualization

Figure 78 is not a pretty picture! The network plots in *igraph* have a wide set of parameters we can set. These include node options (starting with *vertex*) and edge options (starting with *edge*). A list of selected options is included below, but we can also check out by typing *?igraph.plotting* at the console prompt for more information.

1. For the *nodes*, the command is with *vertex.xxxx*, where *xxxx* are options, such as *color*, *label*, etc.
2. For the *edges*, the command is with *edge.xxxx*, where *yyyy* are options, such as *color*, *label*, etc.
3. Other commands are *margin*, *frame*, *palette*, *resalce*, etc.

Plot with curved edges (edge.curved=.1) and reduce arrow size:

- We can set the node and edge options in two ways.
  - first one is to specify them in the plot() function, as we do below.
- Note that using curved edges will allow you to see multiple links between two nodes (e.g. links going in either direction or multiplex links)

```
plot(gm, edge.arrow.size=.4, edge.curved=.1)
```

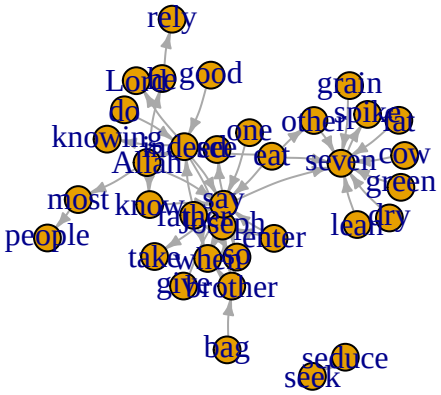


Figure 79: Adjust some edge parameters

Plot output is in Figure 79.

Set edge color to red, the node color to yellow:

- We can set colors by name or hex code.
- Replace the vertex label with the node names stored in "Label"

```
plot(gm, edge.arrow.size=.2, edge.color="red",
      vertex.color="yellow", vertex.frame.color="#ffffff",
      vertex.label=V(gm)$Label, vertex.label.color="black")
```

Plot output is in Figure 80.

Compute node degrees (number of links) and use that to set node size:

```
deg <- degree(gm, mode="all")
V(gm)$size <- deg
plot(gm, vertex.label=V(gm)$Label)
```

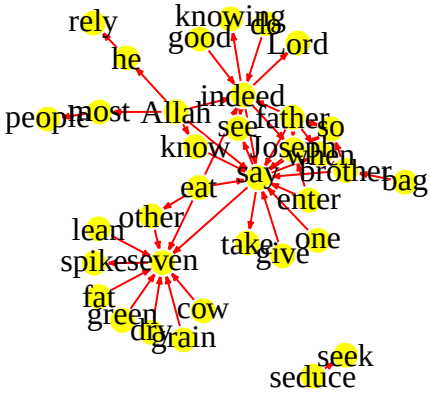


Figure 80: Adjust colors

Figure 81: Adjust node size based on its degree

Plot output is in Figure 81.

```
deg <- degree(gm, mode="all")
V(gm)$size <- deg
plot(gm, vertex.size=igraph::degree(gm), vertex.label=NA)
```

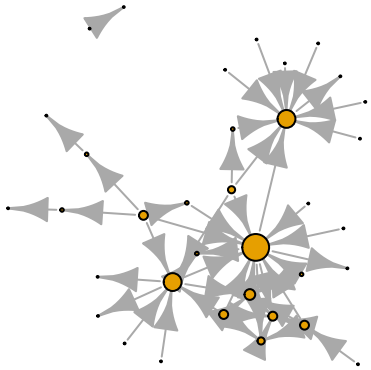


Figure 82: Remove node labels

Plot output is in Figure 82.

Set edge width based on number of co-occurrences:

```
E(gm)$cooc[1:10]
E(gm)$width <- E(gm)$cooc
E(gm)$width[1:10]
plot(gm)
```

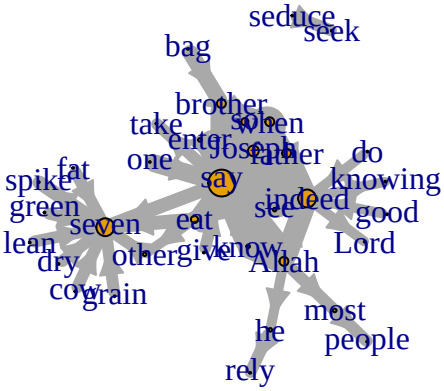


Figure 83: Adjust edge width based on number of co-occurrences

Plot output is in Figure 83.

We change the arrow size and edge color. We introduce and set the network layout in the next code. We will show other layouts later.

```
#change arrow size and edge color:
E(gm)$arrow.size <- .2
E(gm)$edge.color <- "deeppink2"
graph_attr(gm, "layout") <- layout_with_fr
plot(gm)
```

Figure 84: Adjust edge color and set layout

Plot output is in Figure 84.

Sometimes, especially with semantic networks, we plot only the labels of the nodes:

```
plot(gm, vertex.shape="none", vertex.label=V(gm)$name,
      vertex.label.font=2, vertex.label.color="blue",
      vertex.label.cex=.7, edge.color="gray85")
```

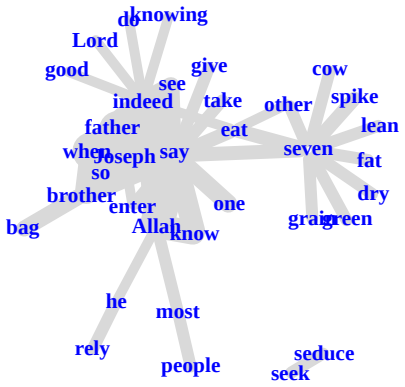


Figure 85: Adjust nodes highlighting only labels

Plot output is in Figure 85.

Color edges of graph based on their source node color:

- Get the starting node for each edge with the *ends() igraph* function.
  - returns the start and end vertex for edges listed in the *es* parameter.
  - names parameter controls whether the function returns edge names or IDs.

```
edge.start <- ends(gm, es=E(gm), names=F)[,1]
edge.col <- V(gm)$color[edge.start]
plot(gm, edge.color=edge.col, edge.curved=.1)
```

Plot output is in Figure 86.

Color edges of graph based on the number of co-occurrences:

Figure 86: Adjust edge color based on the source node

```
E(gm)[cooc <= 5]$color <- "red"
E(gm)[cooc > 5 & cooc <= 10]$color <- "yellow"
E(gm)[cooc > 10]$color <- "green"
plot(gm)
```

Figure 87: Adjust edge color based on formula

Plot output is in Figure 87.

```
plot(gm, vertex.size=5,
     layout=layout.fruchterman.reingold,
     vertex.label=NA)
```

Plot output is in Figure 88.

5.3.3 Graph layouts

Change the graph layout according to some algorithms already implemented in *igraph*. To adjust the figure we just have to assign a new layout:

- layout_with_dh
- layout_with_fr

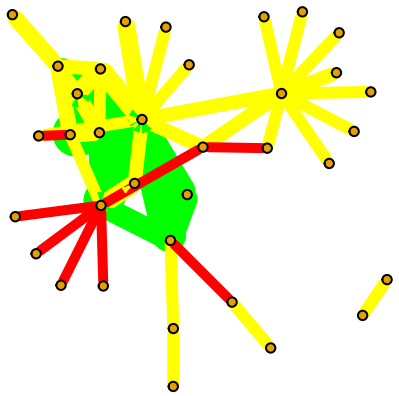


Figure 88: Adjust layout and remove label

- layout_with_kk
- layout_with_sugiyama

```
l<-layout_with_dh(gm)
plot(gm, vertex.color=vertex_attr(gm)$cor,
      vertex.size=igraph::degree(gm),
      edge.width=(edge_attr(gm)$weight)/100,
      edge.color="grey50",
      edge.curved=0.5,
      layout=l)
```

Figure 89: Using dh layout

Plot output is in Figure 89.

Other cool layouts:

```
plot(gm, vertex.color=vertex_attr(gm)$cor,vertex.label=NA,
     vertex.size=igraph::degree(gm),
     edge.width=(edge_attr(gm)$weight)/100,
     edge.color="grey50",
     edge.curved=0.3,
     layout=layout_in_circle, main="layout_in_circle")
```

Plot output is in Figure 90.

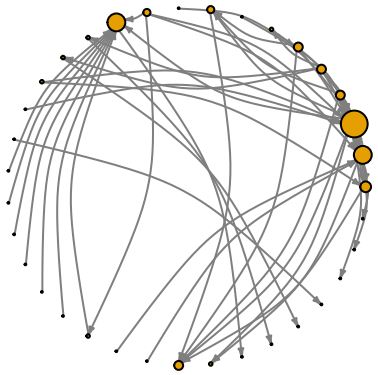


Figure 90: Layout in circle

```
plot(gm, vertex.color=vertex_attr(gm)$cor,
     vertex.size=igraph::degree(gm),
     edge.width=(edge_attr(gm)$weight)/100,
     edge.color="grey50",
     layout=layout_as_tree, main="layout_as_tree")
```

Figure 91: Layout as tree

Plot output is in Figure 91.

```
plot(gm, vertex.color=vertex_attr(gm)$cor,
     vertex.size=igraph::degree(gm),
     edge.width=3*(edge_attr(gm)$weight)/100,
     edge.color="grey50",
     layout=layout_as_star, main="layout_as_star")
```

Plot output is in Figure 92.

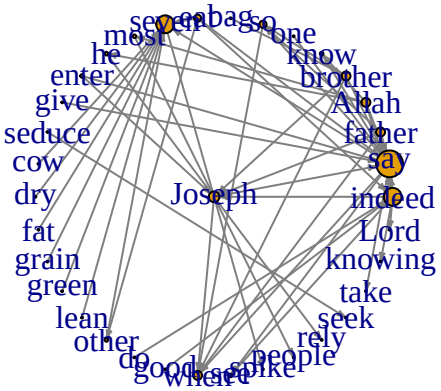


Figure 92: Layout as star

Additional layouts:

- Use a simple program
- List the available layouts and then grep to select

```
layouts <- grep("^layout_", ls("package:igraph"),
                value=TRUE)[-1]
layouts <- layouts[!grepl("bipartite|merge|norm|sugiyama|tree",
                          layouts)]
par(mfrow=c(3,3), mar=c(1,1,1,1))
for (layout in layouts) {
      print(layout)
      l <- do.call(layout, list(gm))
      plot(gm, edge.arrow.mode=0, edge.width=1, layout=l,
           main=layout) }
```

Fruchterman-Reingold:

- One of the most used force-directed layout algorithms out there.
- Force-directed layouts try to get a nice-looking graph where edges are similar in length and cross each other as little as possible.
- They simulate the graph as a physical system.
  - Nodes are electrically charged particles that repel each other when they get too close.
  - Edges act as springs that attract connected nodes closer together.
  - Thus nodes are evenly distributed through the chart area, and the layout is intuitive in that nodes that share more connections are closer to each other.
- The disadvantage of these algorithms is that they are rather slow and therefore less often used in graphs larger than about 1000 nodes.
- With force-directed layouts, you can use the *niter* parameter to control the number of iterations to perform.
  - Default is set at 500 iterations.
  - Lower that number for large graphs to get results faster and check if they look reasonable.

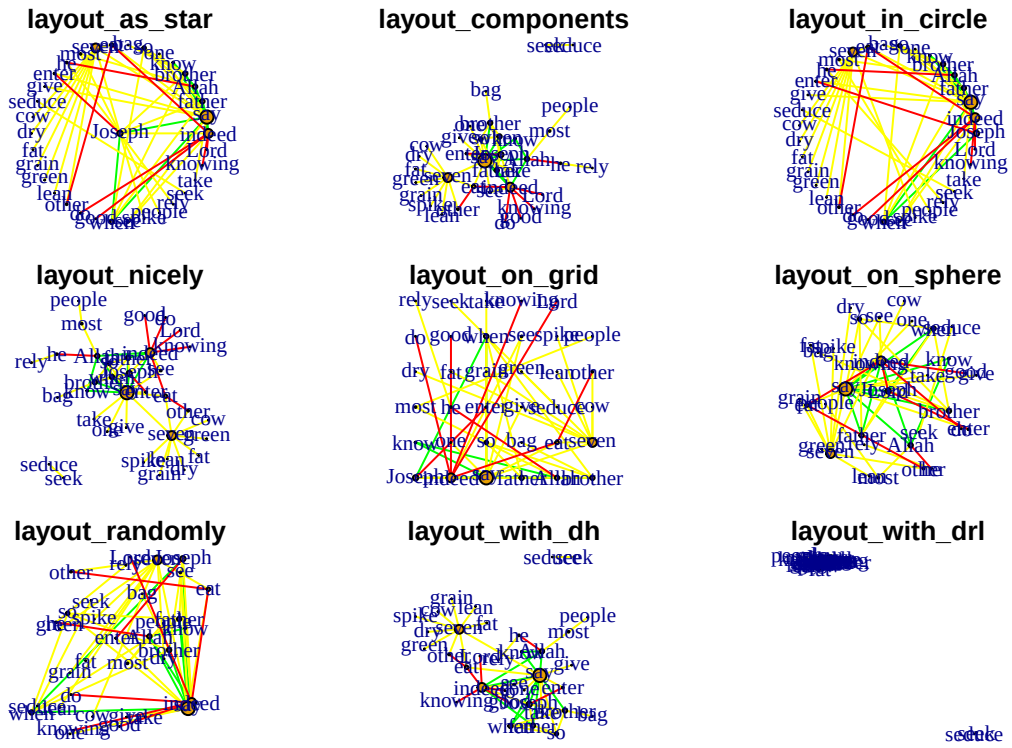


Figure 93: Additional layouts

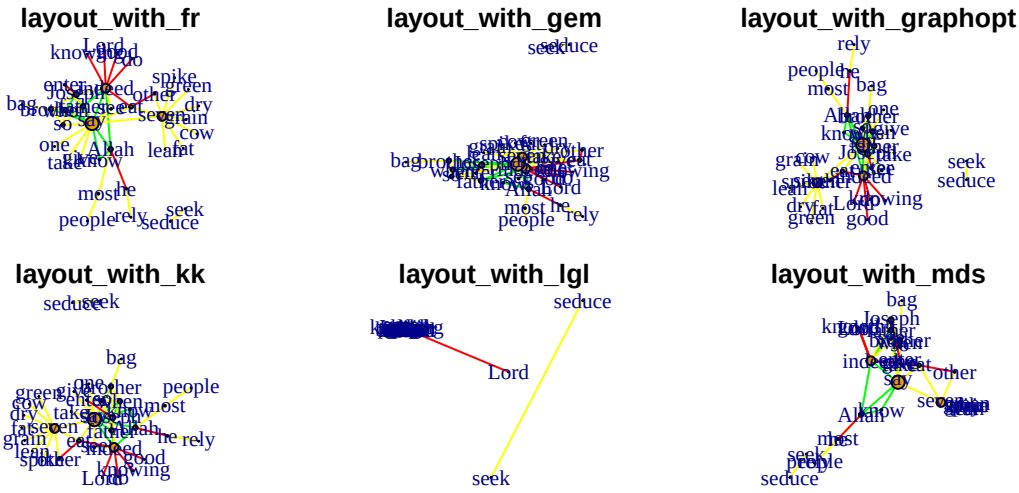


Figure 94: Additional layouts

- The layout can also interpret edge weights. You can set the "weights" parameter which increases the attraction forces among nodes connected by heavier edges.

```
l <- layout_with_fr(gm)
plot(gm, layout=l)
```

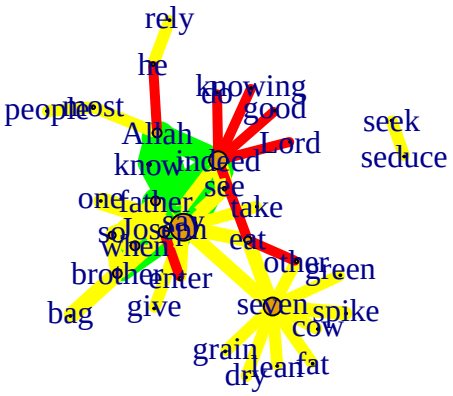


Figure 95: Using fr layout

Plot output is in Figure 95.

```
l <- layout_with_fr(gm, niter=50)
plot(gm, layout=l)
```

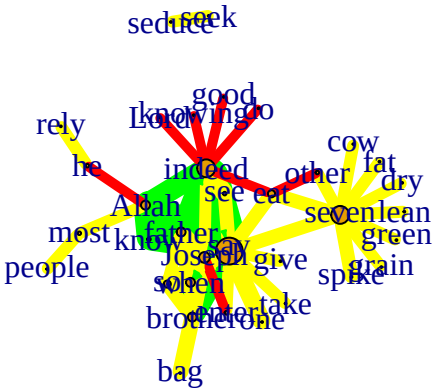


Figure 96: Using fr layout with 50 iterations

Plot output is in Figure 96.

```
ws <-  c(1, rep(100, ecount(gm)-1))
lw <- layout_with_fr(gm, weights=ws)
plot(gm, layout=lw)
```

Plot output is in Figure 97.

Fruchterman-Reingold Non-Deterministic:

- Fruchterman-Reingold layout is not deterministic

Figure 97: Using fr layout different weights

  - different runs will result in slightly different configurations.
  - Saving the layout in *l* allows us to get the exact same result multiple times, which can be helpful if you want to plot the time evolution of a graph or different relationships – and want nodes to stay in the same place in multiple plots.

- By default, the coordinates of the plots are rescaled to the [-1,1] interval for both x and y.
  - change that with the parameter *rescale=FALSE* and rescale plot manually by multiplying the coordinates by a scalar.
  - use *norm_coords* to normalize the plot with the boundaries you want. This way you can create more compact or spread out layout versions.

```
par(mfrow=c(2,2), mar=c(0,0,0,0))
plot(gm, layout=layout_with_fr)
plot(gm, layout=layout_with_fr)
plot(gm, layout=l)
plot(gm, layout=l)
```

Plot output is in Figure 98.

```
l <- layout_with_fr(gm)
l <- norm_coords(l, ymin=-1, ymax=1, xmin=-1, xmax=1)

par(mfrow=c(2,2), mar=c(0,0,0,0))
plot(gm, rescale=F, layout=l*0.4)
plot(gm, rescale=F, layout=l*0.6)
plot(gm, rescale=F, layout=l*0.8)
plot(gm, rescale=F, layout=l*1.0)
```

Plot output is in Figure 99.

Kamada Kawai:

- Another popular force-directed algorithm that produces nice results for connected graphs.
- Like Fruchterman Reingold, it attempts to minimize the energy in a spring system.

```
l <- layout_with_kk(gm)
plot(gm, layout=l)
```

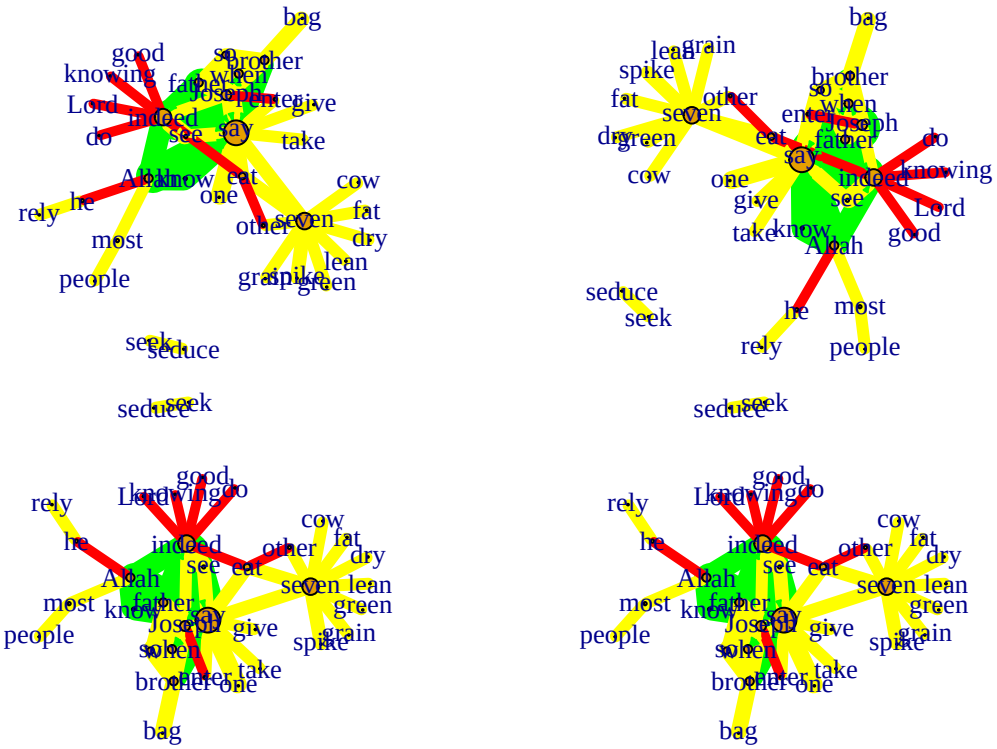


Figure 98: Each fr layout call with different outcomes

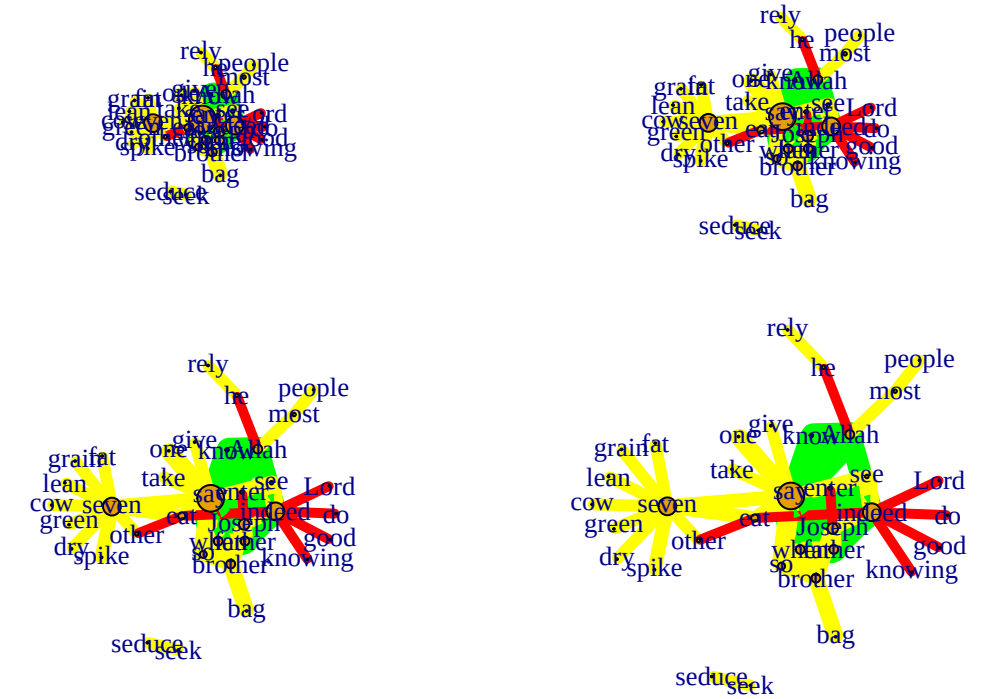


Figure 99: Using fr layout call with manual rescaling

Figure 100: Using kk layout

Plot output is in Figure 100.

Graphopt:

- A nice force-directed layout implemented in igraph that uses layering to help with visualizations of large networks.
- The available graphopt parameters can be used to change the mass and electric charge of nodes, as well as the optimal spring length and the spring constant for edges. The parameter names are:
  - charge (defaults to 0.001),
  - mass (defaults to 30),
  - spring.length (defaults to 0), and
  - spring.constant (defaults to 1).
- Tweaking those can lead to considerably different graph layouts.

```
l <- layout_with_graphopt(gm)
plot(gm, layout=l)
```

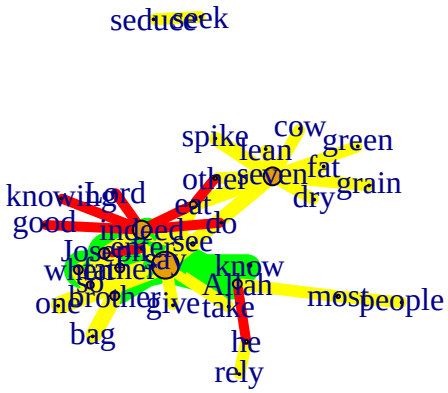


Figure 101: Using graphopt layout

Plot output is in Figure 101.

The charge parameter below changes node repulsion:

```
l1 <- layout_with_graphopt(gm, charge=0.02)
l2 <- layout_with_graphopt(gm, charge=0.00000001)

par(mfrow=c(1,2), mar=c(1,1,1,1))
plot(gm, layout=l1)
plot(gm, layout=l2)
```

Figure 102: Using graphopt layout with different charge

Plot output is in Figure 102.

5.3.4 Graph algorithms

LGL algorithm:

- Meant for large, connected graphs.
- Here you can also specify a root, a node that will be placed in the middle of the layout.

```
par(mfrow=c(1,2), mar=c(1,1,1,1))
plot(gm, layout=layout_with_lgl, root = 1)
plot(gm, layout=layout_with_lgl, root = 5)
```

Plot output is in Figure 103.

The MDS (multidimensional scaling) algorithm:

- Tries to place nodes based on some measure of similarity or distance between them. More similar nodes are plotted closer to each other.
- By default, the measure used is based on the shortest paths between nodes in the network.
- We can change that by using our own distance matrix (however defined) with the parameter dist.
- MDS layouts are nice because positions and distances have a clear interpretation.
- The problem with them is visual clarity: nodes often overlap, or are placed on top of each other.

```
plot(gm, layout=layout_with_mds)
```

Plot output is in Figure 104.

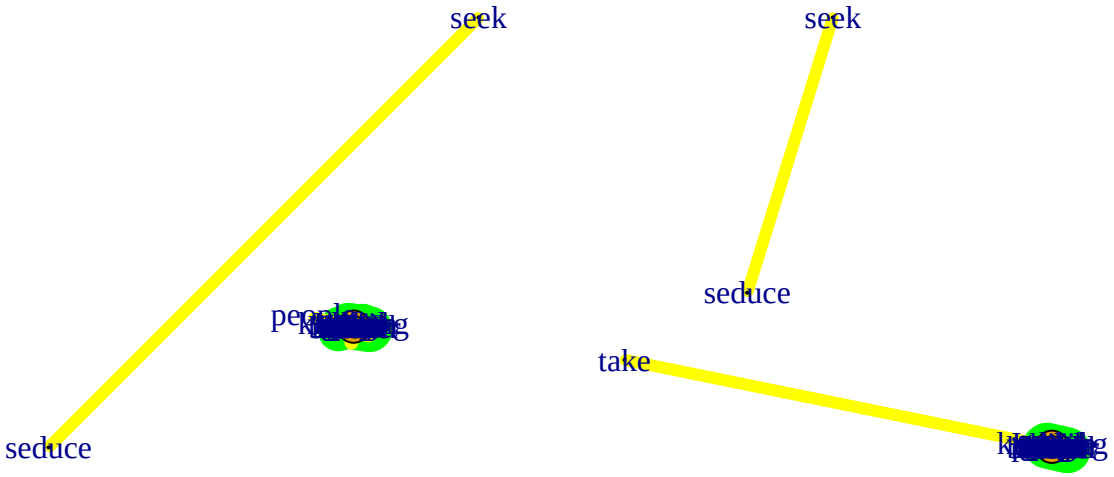


Figure 103: Using lgl layout with different roots

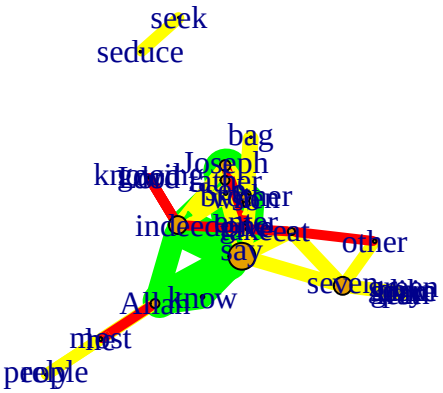


Figure 104: Using mds layout

5.3.5 Graph analysis

- *igraph* has several built-in functions to analyze graph properties.
- We can check for communities or clusters with a choice of algorithms.
  - cluster_spinglass()
  - cluster_optimal()
  - cluster_walktrap()
- We will calculate modularity using the cluster_spinglass function, which uses simulated annealing for optimization:
- Some of the clustering functions give error results for networks with unconnected nodes.
  - It is obvious from the previous plots that the nodes **"seek"** and **"seduce"** are unconnected with the rest.
  - The code below shows how to use the delete_vertices() function.

```
gm1 <- delete_vertices(gm, "seek")
gm1 <- delete_vertices(gm1, "seduce")
gmc <- cluster_spinglass(gm1)
str(gmc) #evaluating output
gmc$membership #checking to which module each node belongs to
plot(gm1, vertex.color=vertex_attr(gm1)$cor,
     vertex.size = 3*degree(gm1),
     edge.width = edge_attr(gm1)$coor,
     edge.color = "grey50",
     edge.curved = 0.3,
     layout = layout_with_kk,
     mark.groups = gmc,
     mark.border=NA)
```

Figure 105: Communities or clusters in tutorial word network

Plot output is in Figure 105.

Calculate and plot the shortest path between two nodes:

- To emphasize the path between them, we have to create a new color vector for interactions (edges) the same way we did before for the nodes/vertices:

```
short.path <- shortest_paths(gm, from = "Joseph", to = "seven", output = "both")
short.path
# creating a color vector for all the edges with the color "grey80";
```

```
# the ecount fucntion tells you how many edges the network has
ecol <- rep("gray80", ecount(gm))
# coloring in red only the path in which we are interested,
# that we calculed using the shortest_paths function.
ecol[unlist(short.path$epath)] <- "red"
l <- layout_with_kk(gm)
plot(gm, vertex.color=vertex_attr(gm)$cor,
     vertex.size=igraph::degree(gm),
     edge.width=2,
     edge.color=ecol,
     layout=l)
```

Figure 106: Path from one node to another

Plot output is in Figure 106.

5.3.6 Using ggraph

- The *ggplot2* package and its extensions are known for offering a more meaningfully structured and advanced way to visualize data in R.
- In *ggplot2*, we can select from a variety of visual building blocks and add them to our graphics one by one, a layer at a time.
- The *ggraph* package takes this principle and extends it to network data. In this section, we will only cover the basics.
- We can use our *igraph* objects directly with the *ggraph* package.
- The following code gets the data and adds separate layers for nodes and links.

```
library(ggraph)
lay = create_layout(gm, layout = "kk")
ggraph(lay) +
  geom_edge_link() +
  geom_node_point() +
  theme_graph()
```

Plot output is in Figure 107.

Add node names and edge type:

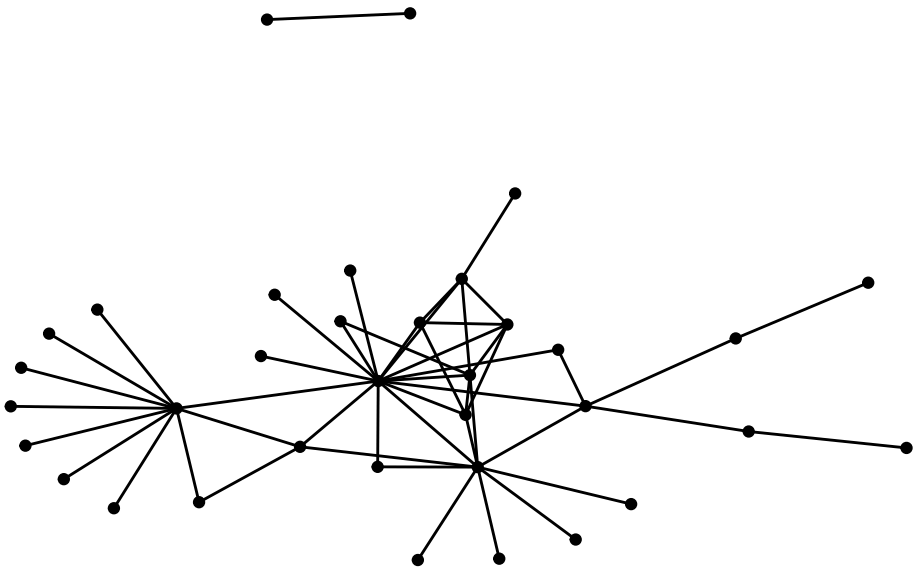


Figure 107: Using ggraph with minimal parameters set

```
ggraph(lay) +
    geom_edge_link() +
    geom_node_point() +
    geom_node_text(aes(label = name), repel=TRUE) +
    geom_edge_link(aes(color = cooc))
```

Plot output is in Figure 108.

Plot node degree as node size (and alpha):

```
ggraph(lay) +
    geom_node_point() +
    geom_node_text(aes(label = name), repel=TRUE) +
    geom_node_point(aes(size = degree(gm), alpha = degree(gm)),
                    color = "#de4e96") +
    geom_edge_link(aes(color = cooc))
```

Plot output is in Figure 109.

Plot edge weight as edge alpha:

```
ggraph(lay) +
  geom_edge_link(aes(alpha = cooc)) +
  geom_node_point()
```

Plot output is in Figure 110.

Layouts in ggraph:

- Some network layouts are familiar from *igraph*:
  - 'star', 'circle', 'grid', 'sphere', 'kk', 'fr', 'mds', 'lgl', etc.

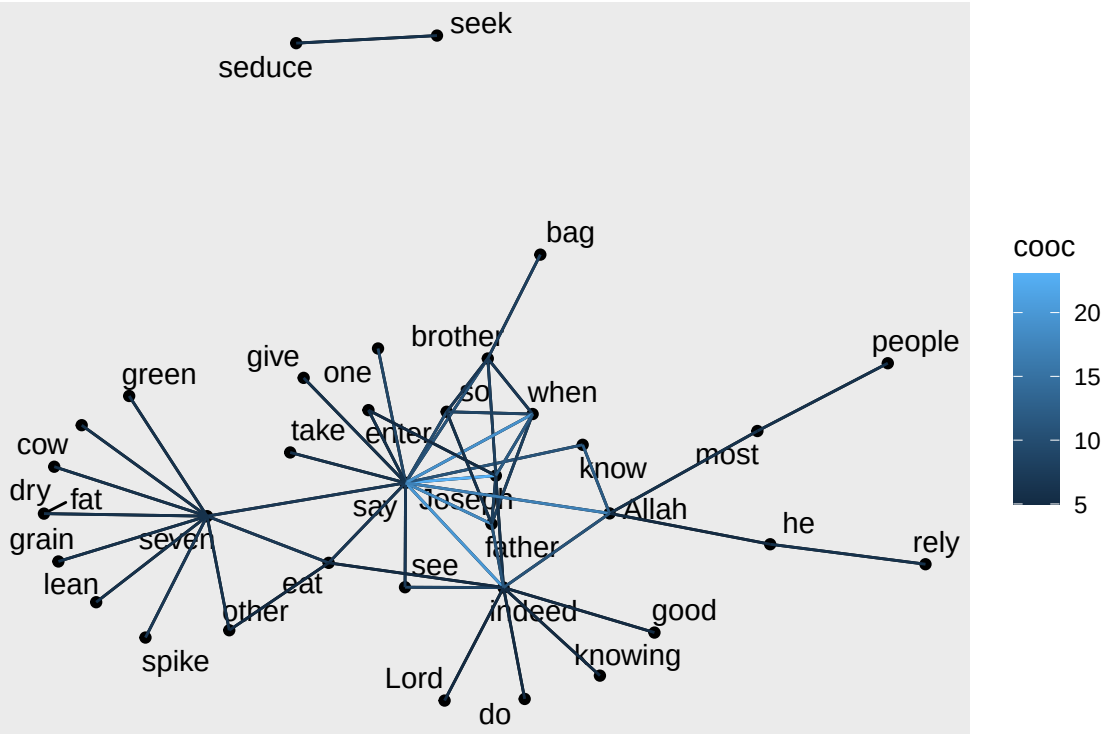


Figure 108: ggraph with node and edge parameters set

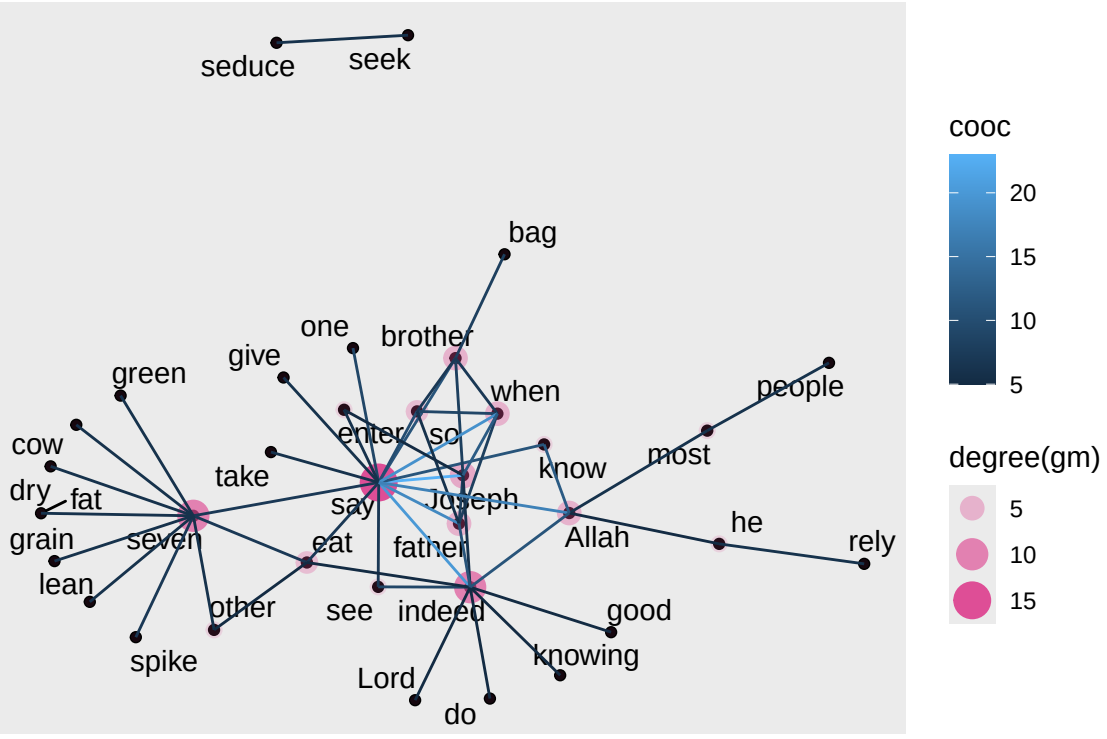


Figure 109: ggraph with node size and alpha based on degree

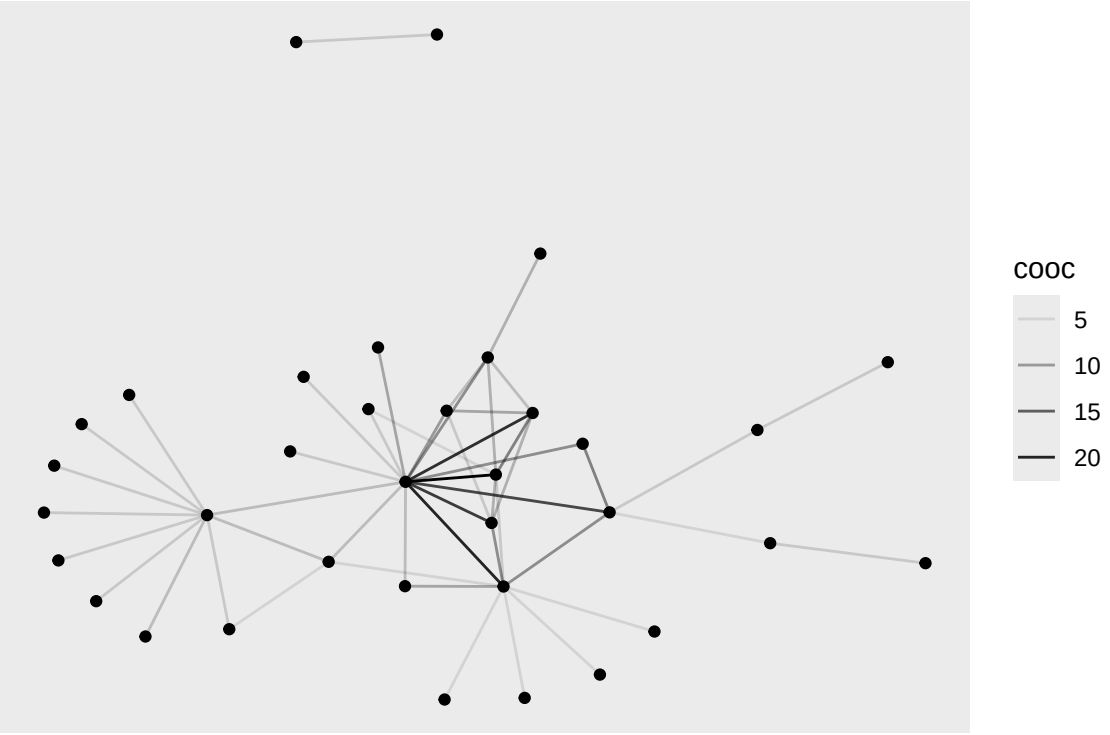


Figure 110: ggraph with edge width and alpha based on number of co-occurrences

- We can use geom_edge_link() for straight edges, geom_edge_arc() for curved ones, and geom_edge_fan() when we want to make sure any overlapping multiplex edges will be fanned out.
- We can set visual properties for the network plot by using key function parameters.
  - Nodes have color, fill, shape, size, and stroke.
  - Edges have color, width, and linetype.
  - alpha parameter controls transparency.

```
ggraph(gm, layout="kk") +
  geom_edge_link() +
  ggtitle("Your Title Here") +  # add title to the plot
  geom_edge_fan(color="gray50", width=0.8, alpha=0.5) +
  geom_node_point(size=2) +
  theme_void()
```

Plot output is in Figure 111.

Different themes:

- As in *ggplot2*, we can add different themes to the plot.
  - For a cleaner look, we can use a minimal or empty theme with theme_minimal() or theme_void().

```
ggraph(gm, layout = 'linear') +
    geom_edge_arc(color = "gold3", width=0.7) +
    geom_node_point(size=3, color="gray50") +
    geom_node_text(aes(label = name), repel=TRUE, size = 3) +
    theme_void()
```

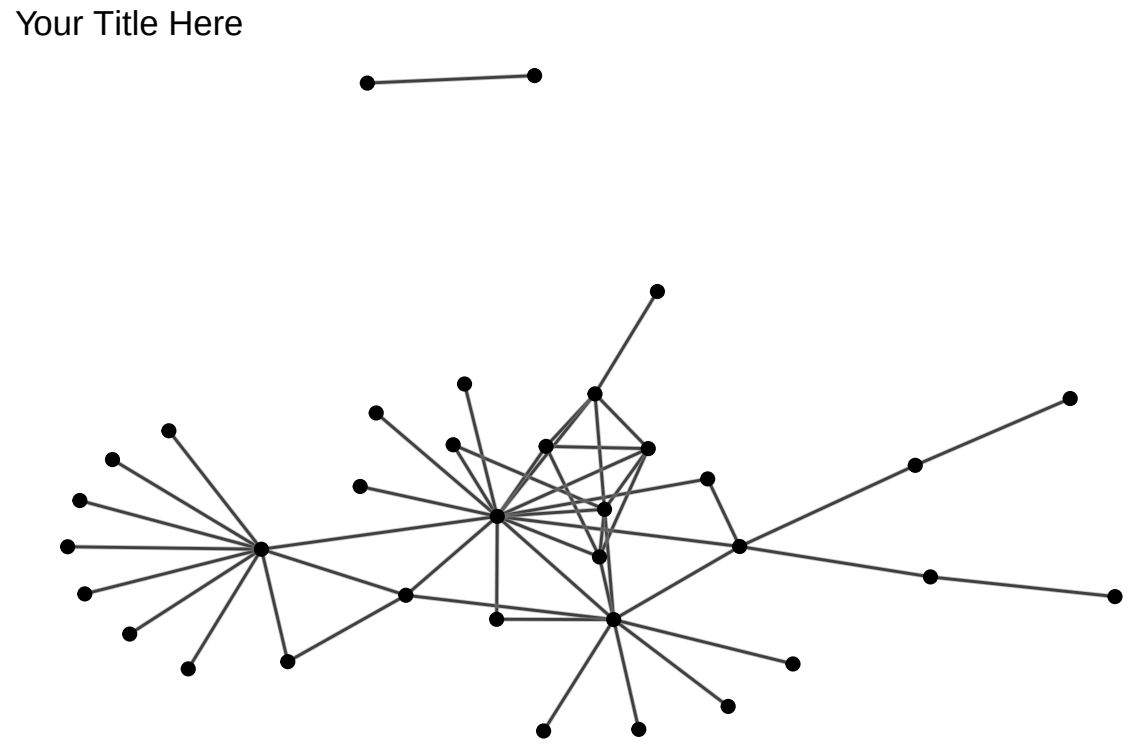


Figure 111: Using ggraph with layout kk

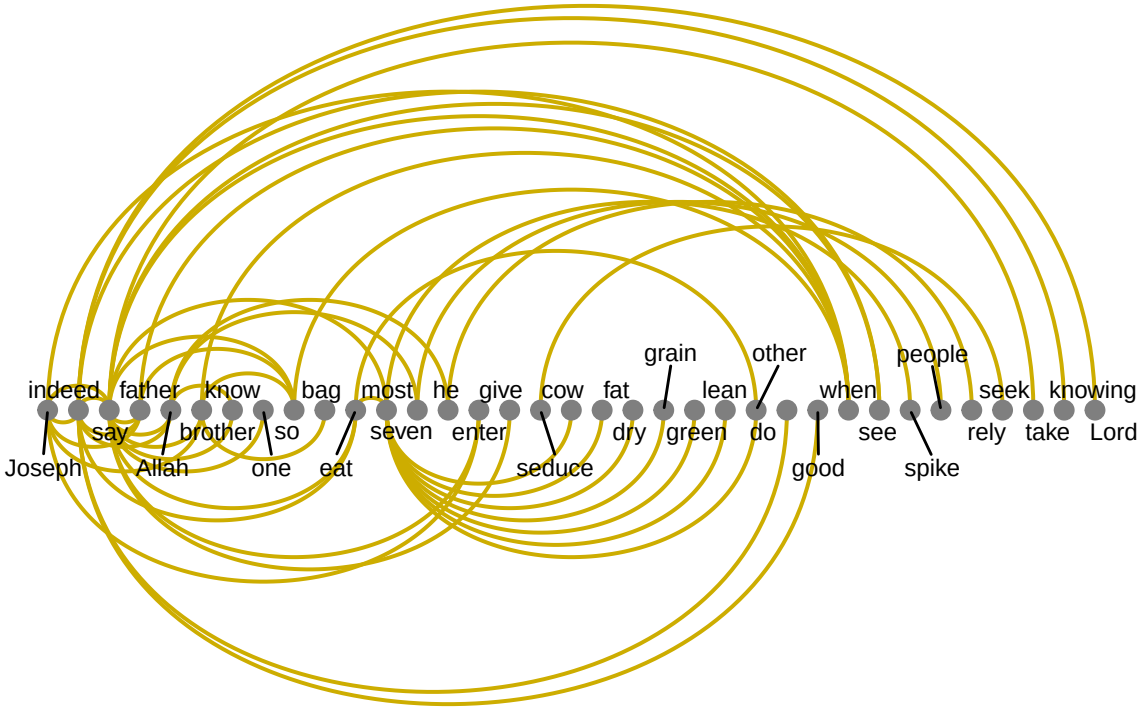


Figure 112: Using ggraph with linear layout and void theme

Plot output is in Figure 112.

Mapping aesthetics:

- The *ggraph* package also uses the traditional *ggplot2* way of mapping aesthetics: specifying which elements of the data should correspond to different visual properties of the graphic.
- It is done using the aes() function that matches visual parameters with attribute names from the data.
- In the code below, the edge attribute type and node attribute audience.size are taken from our data as they are included in the *igraph* object.

```
ggraph(gm, layout="kk") +
  geom_edge_link(aes(color = cooc)) +
  geom_node_point(aes(size = degree(gm))) +
  theme_void()
```

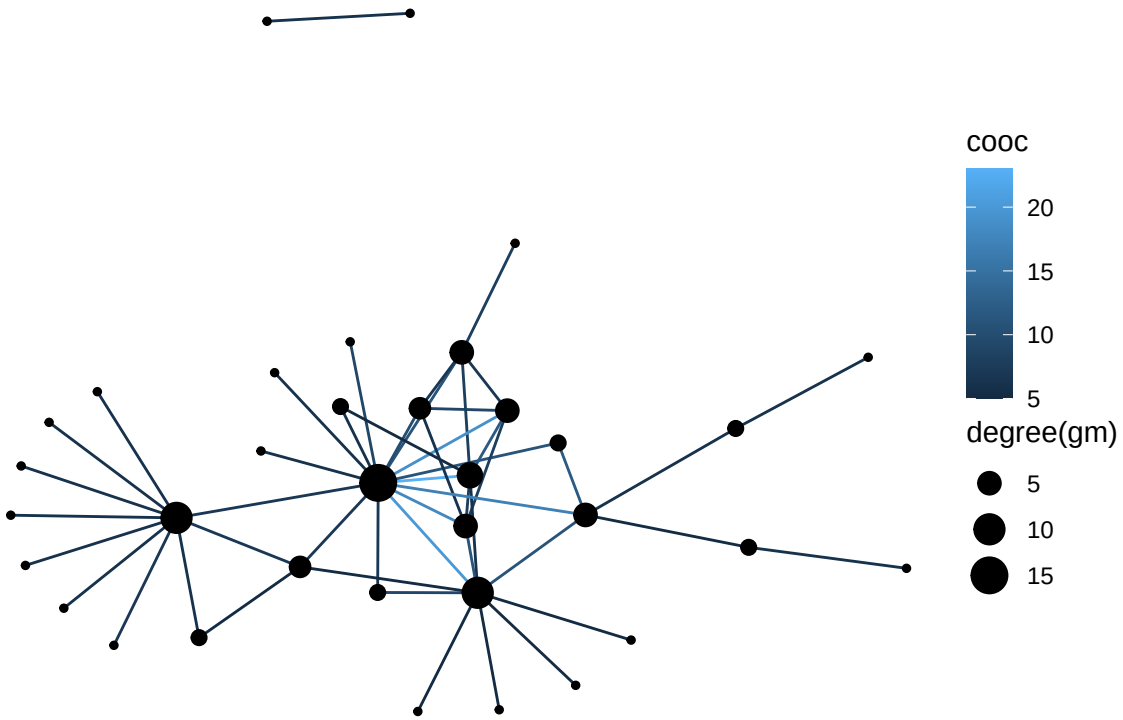


Figure 113: Using ggraph with kk layout and aes setting

Plot output is in Figure 113.

Legends:

- One great thing about *ggplot2* and *ggraph* we can see above is that they automatically generate a legend which makes plots easier to interpret.
- We can add a layer with node labels using geom_node_text() or geom_node_label() which correspond to similar functions in *ggplot2*.

```
ggraph(gm,  layout = 'kk') +
  geom_edge_arc(color="gray", curvature=0.3) +
  geom_node_point(color="#de4e96", aes(size = degree(gm))) +
  geom_node_text(aes(label = name), size=3, color="darkblue",
```

```
                repel=T) +
  theme_void()
```

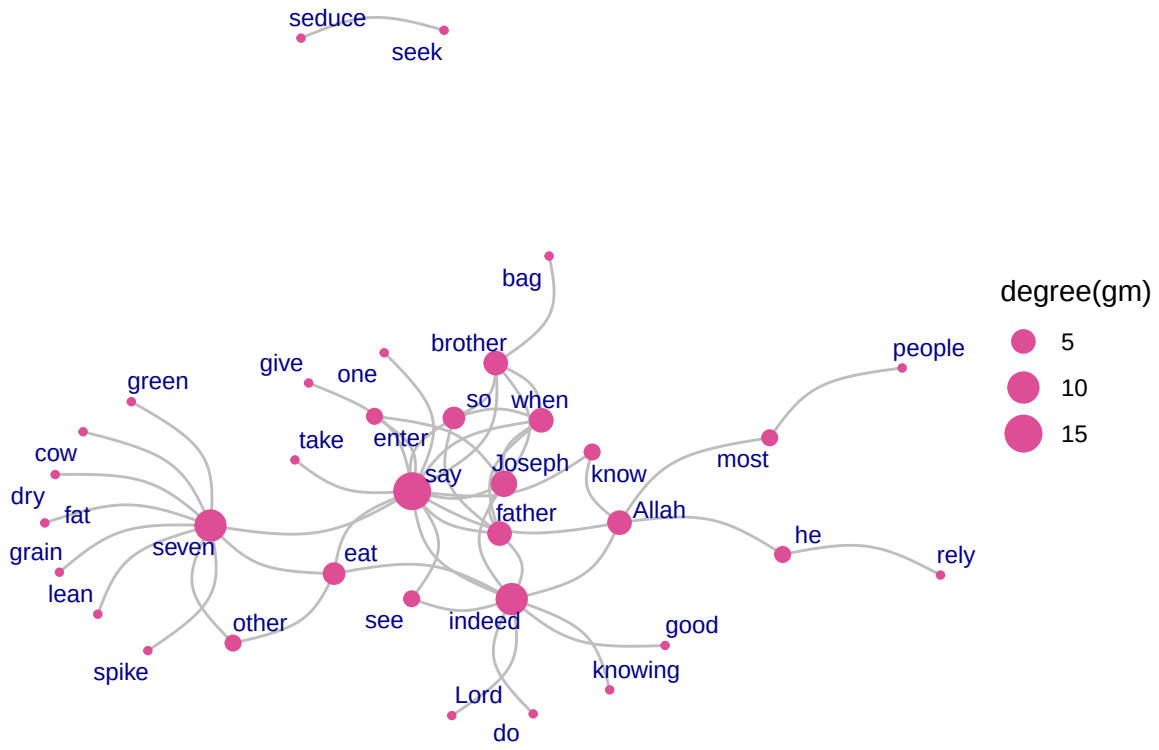


Figure 114: Using ggraph with various parameter settings

Plot output is in Figure 114.

Arc and Coord:

- *ggraph* provides some new layout types and algorithms for our drawing pleasure

```
ggraph(gm, layout = 'linear',  circular = TRUE) +
     geom_edge_arc(aes(color = cooc)) +
     geom_node_point() +
     geom_node_text(aes(label = name), size=3, color="darkblue",
                    repel=T)
```

Plot output is in Figure 115.

Fan:

- Sometimes the graph is not simple, i.e. it has multiple edges between the same nodes.
- Using links is a bad choice here because edges will overlap and the viewer will be unable to discover parallel edges.
- geom_edge_fan() helps here.
  - If there are no parallel edges it behaves like geom_edge_link() and draws a straight line.
  - If parallel edges exist it will spread them out as arcs with different curvature.
  - Parallel edges will be sorted by directionality before plotting so edges flowing in the same direction will be plotted together.

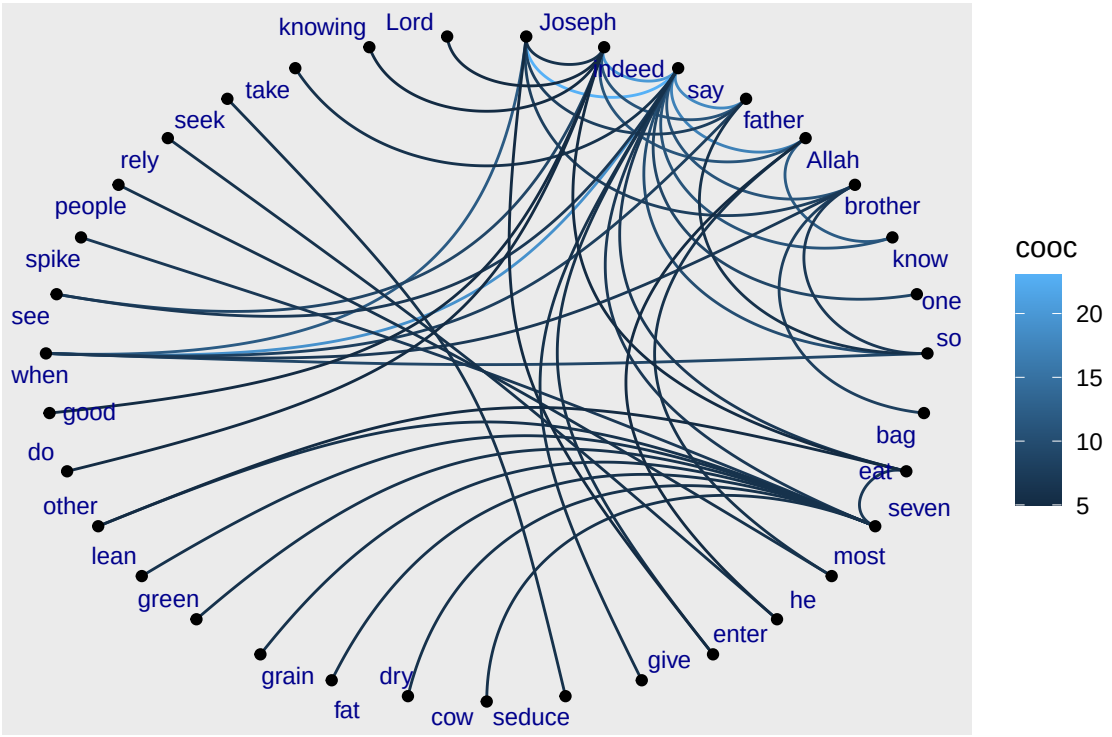


Figure 115: Using ggraph with linear layout

```
ggraph(gm, layout = 'kk') +
     geom_edge_fan(aes(color = cooc, width = cooc)) +
     geom_node_point() +
     geom_node_text(aes(label = name), size=3, color="darkred",
                    repel=T)
```

Plot output is in Figure 116.

Density:

- Consider the case where it is of interest to see which types of edges dominate certain areas of the graph.
- We can color the edges, but edges can tend to get overplotted, thus reducing readability.
- geom_edge_density() lets us add shading to our plot based on the density of edges in a certain area:

```
ggraph(gm, layout = 'kk') +
     geom_edge_density(aes(fill = cooc)) +
     geom_edge_link(alpha = 0.25, color = "steelblue") +
     geom_node_point(color="#de4e96", aes(size = igraph::degree(gm)))
```

Plot output is in Figure 117.

Other ways to represent a network:

- *ggraph* offers several other interesting ways to represent networks, including dendrograms, treemaps, hive plots, and circle plots.
- We show below a simple heatmap of our network matrix.

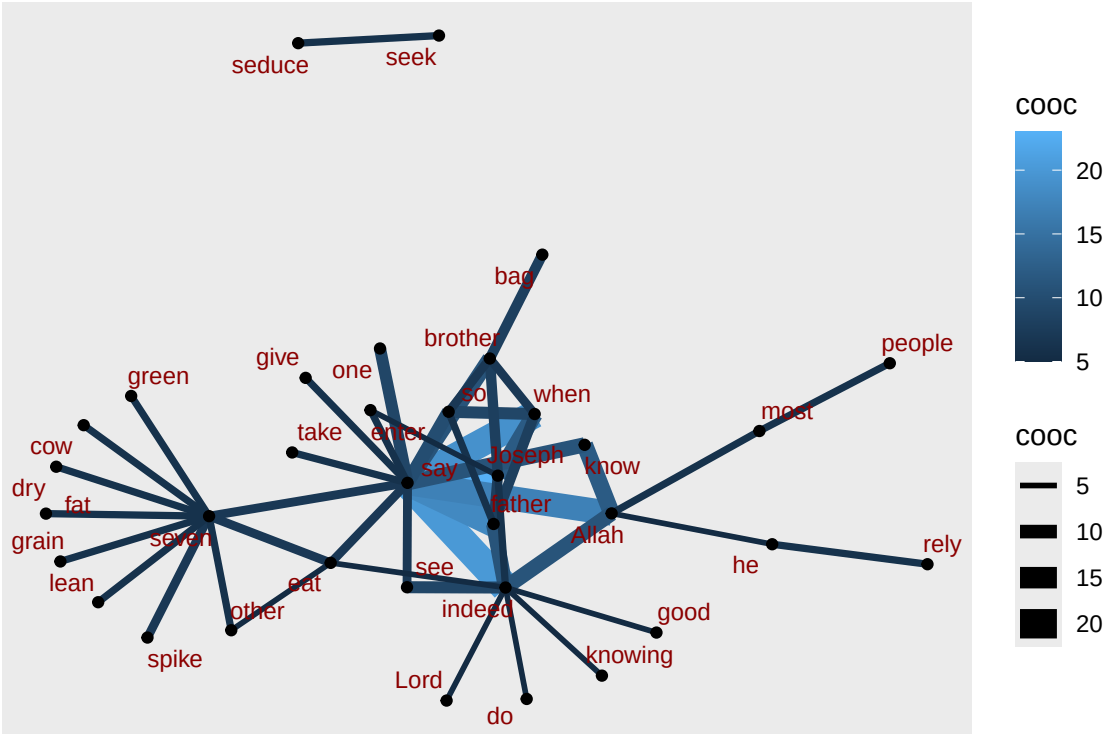


Figure 116: Using ggraph with edge fan

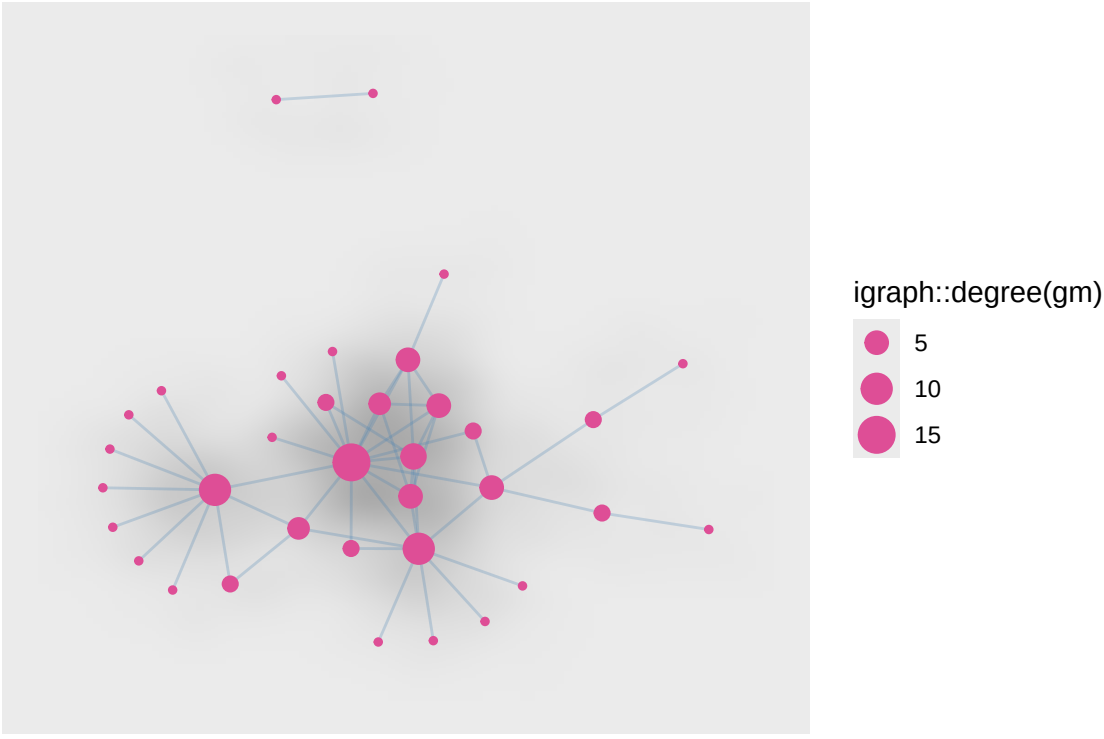


Figure 117: Using ggraph with edge density

```
netm <- get.adjacency(gm, attr="cooc", sparse=F)
colnames(netm) <- V(gm)$name
rownames(netm) <- V(gm)$name
palf <- colorRampPalette(c("yellow", "red"))
heatmap(netm[,17:1], Rowv = NA, Colv = NA, col = palf(100),
        scale="none", margins=c(10,10) )
```

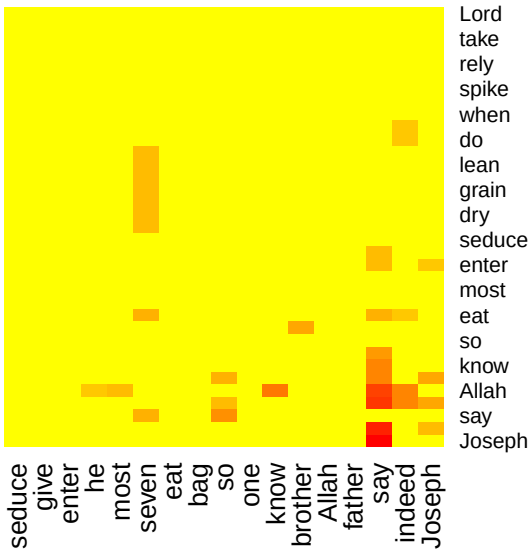


Figure 118: Heatmap of word network

Plot output is in Figure 118.

Plot the degree distribution for our network:

- Depending on what properties of the network or its nodes and edges are most important to us, simple graphs can often be more informative.

```
deg.dist <- degree_distribution(gm, cumulative=T, mode="all")
plot( x=0:max(degree(gm)), y=1-deg.dist, pch=19, cex=1.2, col="orange",
      xlab="Degree", ylab="Cumulative Frequency")
```

Plot output is in Figure 119.

5.4 Fun with network graphs

Let us have some light moments working with network graphs. We have seen the various network layout options, and also for geom_edge_density, geom_edge_link, geom_node_point, geom_node_text. Different combinations of these options can result in interesting and fun ways to represent network graph data.

Digital art is one interesting aspect of visualizing data. We will show an example in this section. For that, we increase the network size of the word co-occurrences in Surah Yusuf and explore.

```
wordnetwork <- head(cooccur, 100)
gm1 <- graph_from_data_frame(wordnetwork)
ggraph(gm1, layout = 'kk') +
```

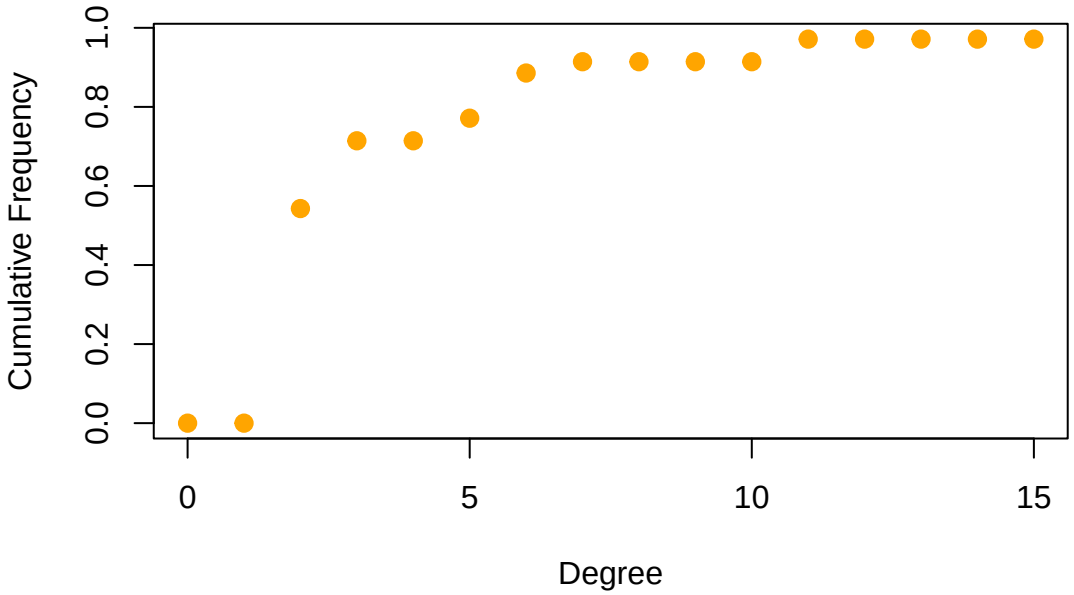


Figure 119: Degree distribution of word network

```
      geom_edge_density(aes(fill = cooc)) +
      geom_edge_link(alpha = 0.7, color = "#57d3e6") +
      geom_node_point(aes(size = degree(gm1)), colour = "#a83268") +
      geom_node_text(aes(label = name), size = 3, repel=TRUE) +
      ggtitle("Network of Top 100 Words")
```

Plot output is in Figure 120.

```
ggraph(gm1, layout = 'kk') +
      geom_edge_density(aes(fill = cooc)) +
      geom_edge_link(alpha = 0.7, color = "#57d3e6") +
      geom_node_point(aes(size = degree(gm1)),
                      colour = "#a83268") +
      ggtitle("Network of Top 100 Words")
```

Plot output is in Figure 121.

We remove the title and legend. We encourage our readers to try different combinations of colors, alphas, and network size. Surah Yusuf is a beautiful Surah. The word co-occurrence network of this Surah looks quite pretty too.

```
ggraph(gm1, layout = 'kk') +
      geom_edge_density(aes(fill = cooc)) +
      geom_edge_link(alpha = 0.7, color = "#57d3e6") +
      geom_node_point(aes(size = degree(gm1)),
                      colour = "#a83268", show.legend = FALSE)
```

Plot output is in Figure 122.

We now use the linear layout with the circular parameter set to TRUE to give a circular effect.

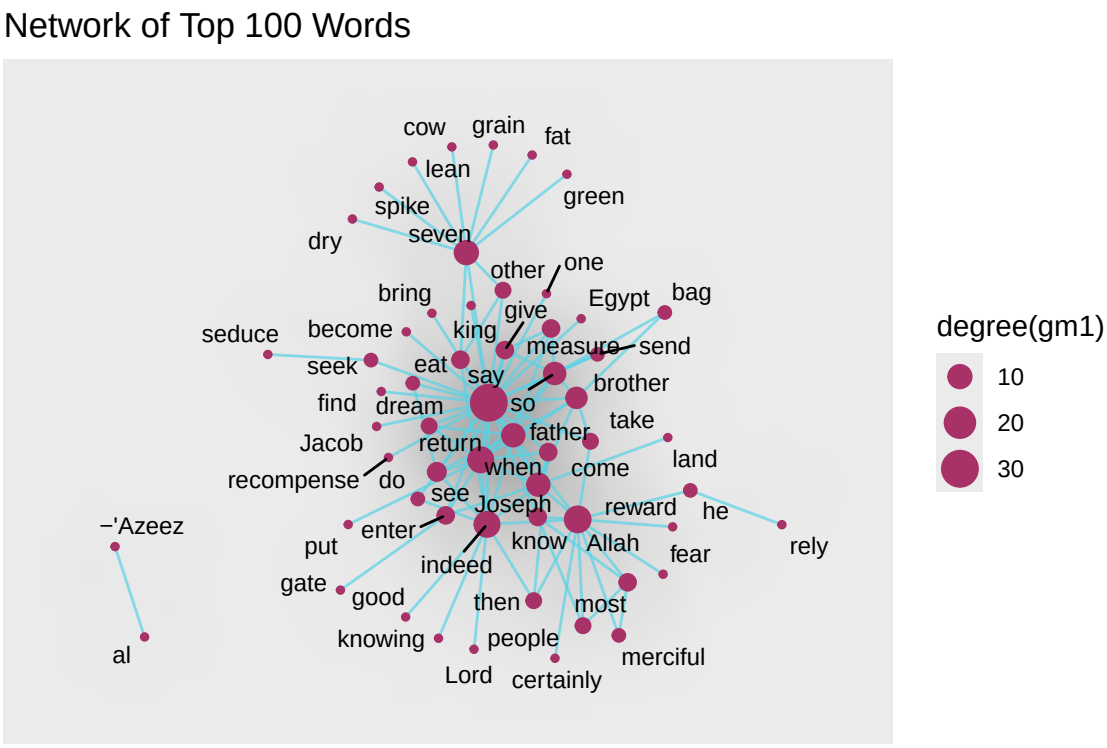


Figure 120: Larger network density with labels

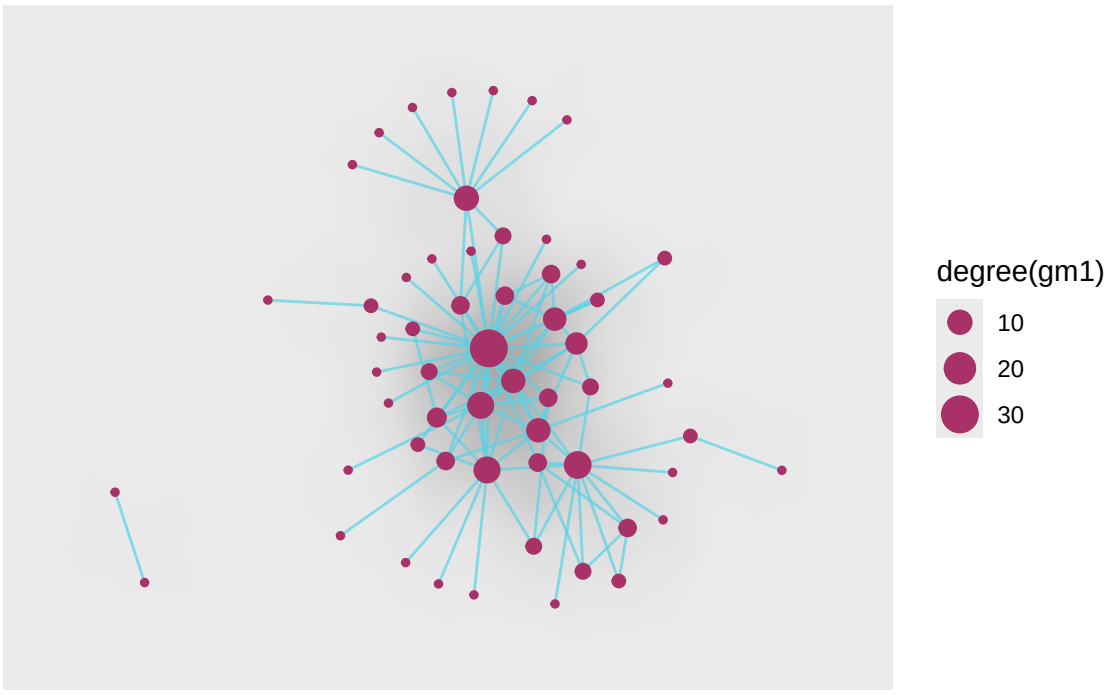


Figure 121: Larger network density without labels

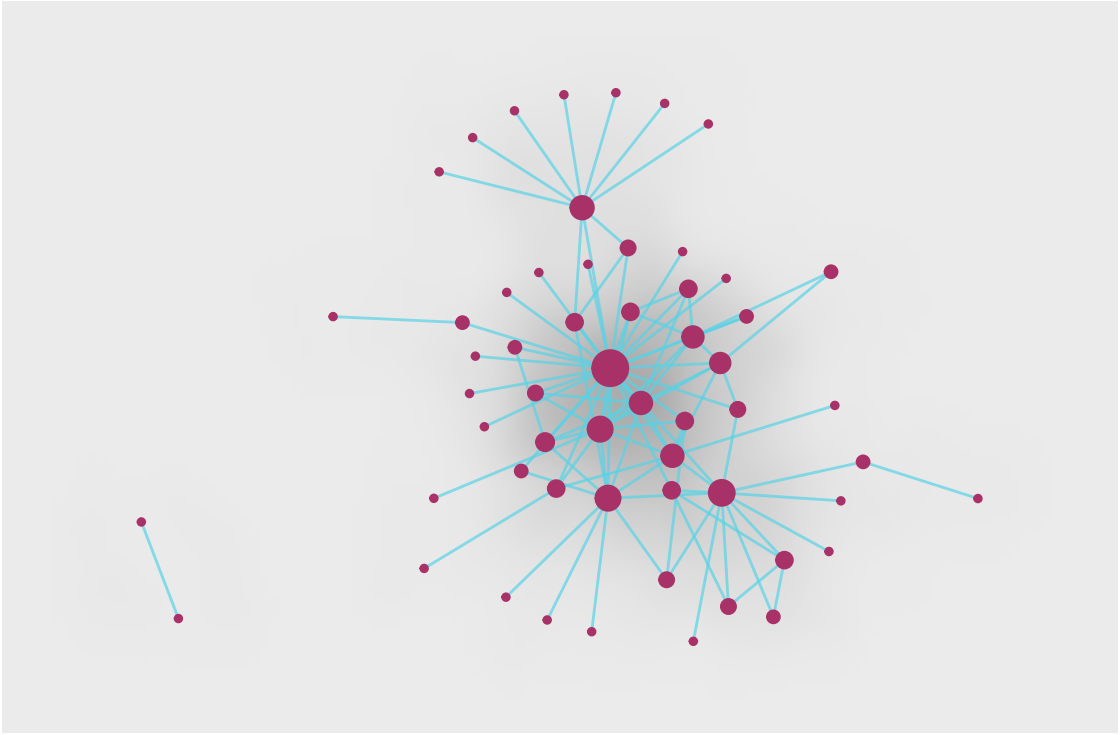


Figure 122: Digital art from Surah Yusuf

```
ggraph(gm1, layout = 'linear', circular = TRUE) +
     geom_edge_arc(color = "#57d3e6", width=0.7) +
     geom_node_point(aes(size = degree(gm1)), alpha = igraph::degree(gm1),
                     colour = "#a83268") +
     geom_node_text(aes(label = name), size = 3, repel=TRUE) +
     theme_void() +
     ggtitle("Network of Top 100 Words")
```

Plot output is in Figure 123.

5.4.1 Working with a bigger graph

We increase the word network for Surah Yusuf to 2000 co-occurrences or edges.

```
wordnetwork <- head(cooccur, 2000)
jg <- graph_from_data_frame(wordnetwork)

ggraph(jg, layout = 'kk') +
      geom_edge_density(aes(fill = cooc)) +
      geom_edge_link(alpha = 0.7, color = "#57d3e6") +
      geom_node_point(aes(size = igraph::degree(jg)),
                      colour = "#a83268", show.legend = FALSE) +
      ggtitle("Data Art of Top 2000 Coccurrences")
```

Plot output is in Figure 124.

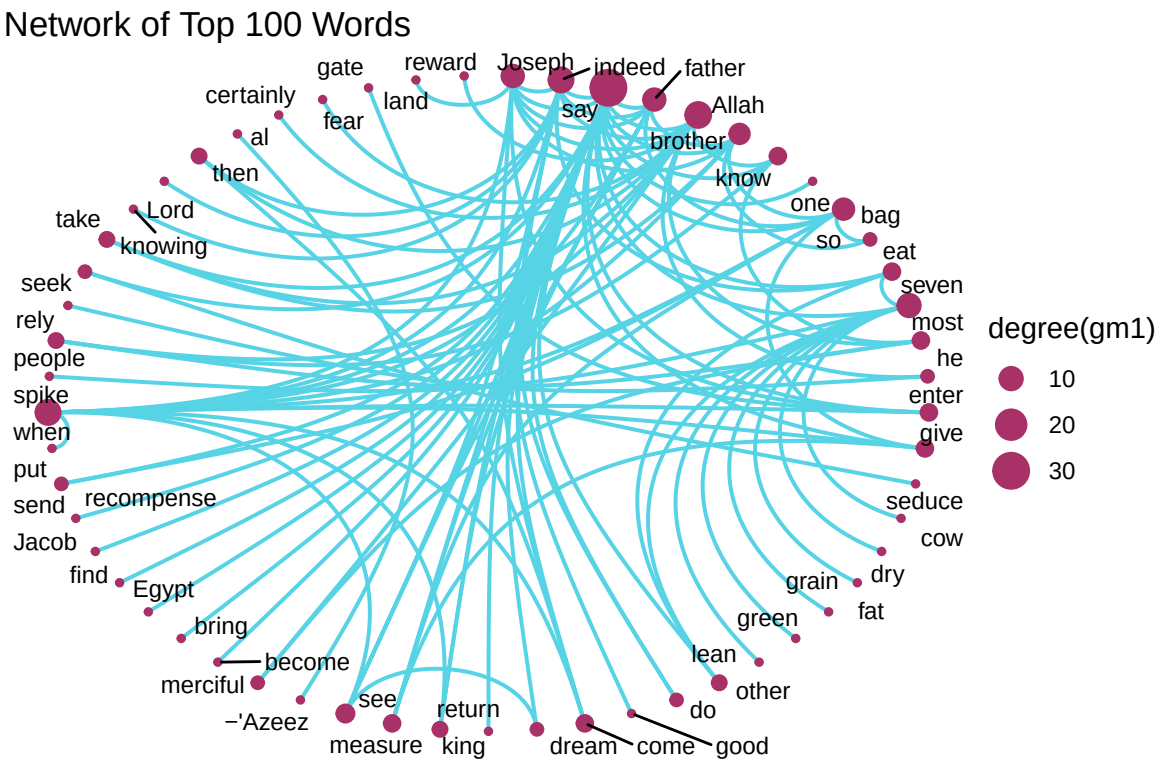


Figure 123: Word network with circular layout

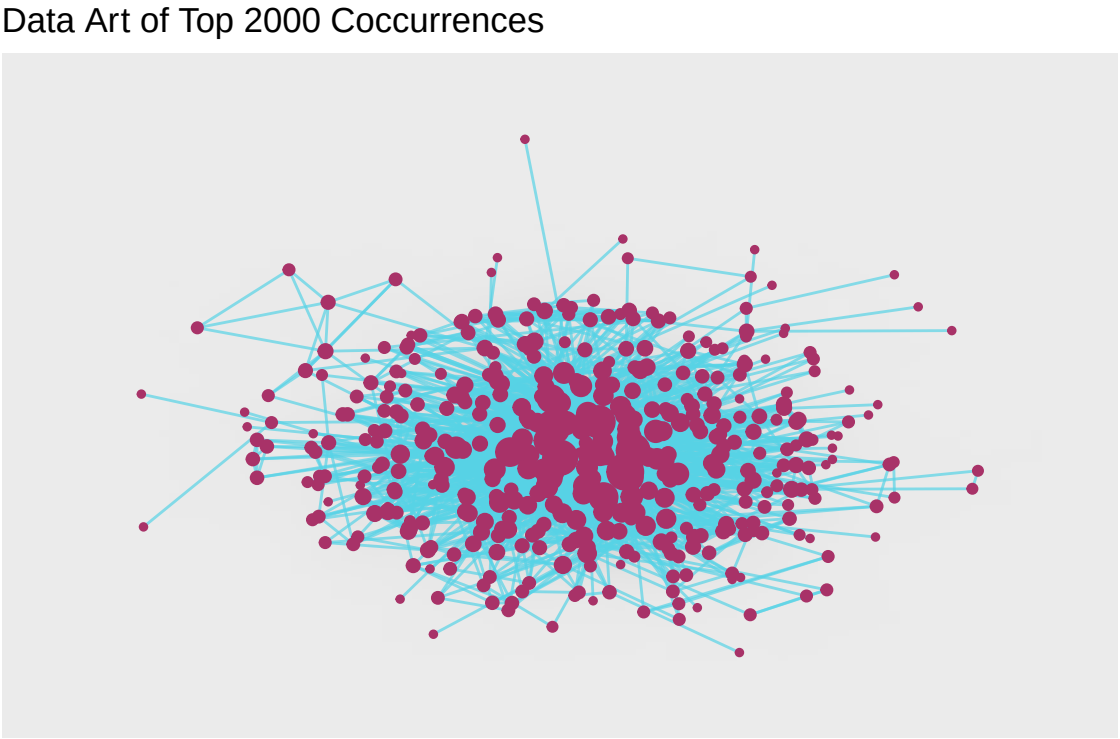


Figure 124: Larger word network with kk layout and edge density

Does the plot above show all the 421 nodes as connected? We can do other analyses.

```
is.connected(jg) # Is it connected?
no.clusters(jg) # How many components?
table(clusters(jg)$csize) # How big are these?
max(degree(jg, mode="in")) # Vertex degree
max(degree(jg, mode="out"))
max(degree(jg, mode="all"))
```

```
# In-degree distribution
plot(degree.distribution(jg, mode="in"), log="xy")
```

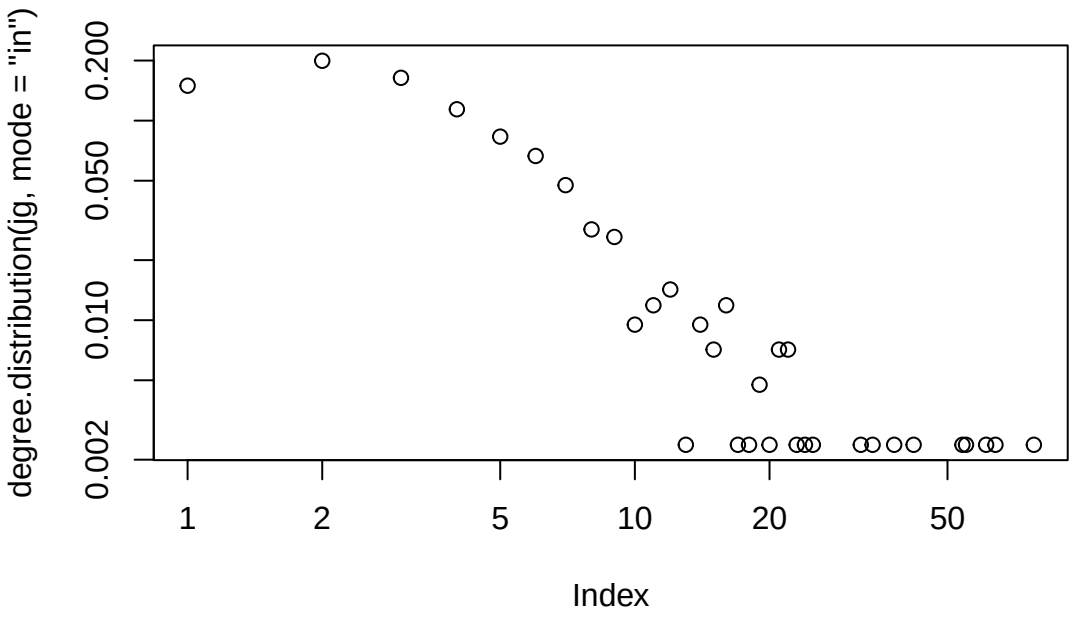


Figure 125: Larger word network with kk layout and edge-in density

Plot output is in Figure 125.

```
# Out-degree distribution
plot(degree.distribution(jg, mode="out"), log="xy")
```

Plot output is in Figure 126.

is.connected(jg) being FALSE says that there are some nodes that are not connected.

```
lay = create_layout(jg, layout = "fr")
ggraph(lay) +
   geom_edge_link() +
   geom_node_point() +
   geom_node_text(aes(label = name), size = 3)
```

The rather ugly Figure 127 shows the 4 nodes. Let us delete and repeat the cluster analysis. But instead of using the delete_vertices() function like in an earlier example, we just use the main component. First, we find the components and then subset the graph based on those components. In *igraph* the largest component is not always the first one with id == 1.

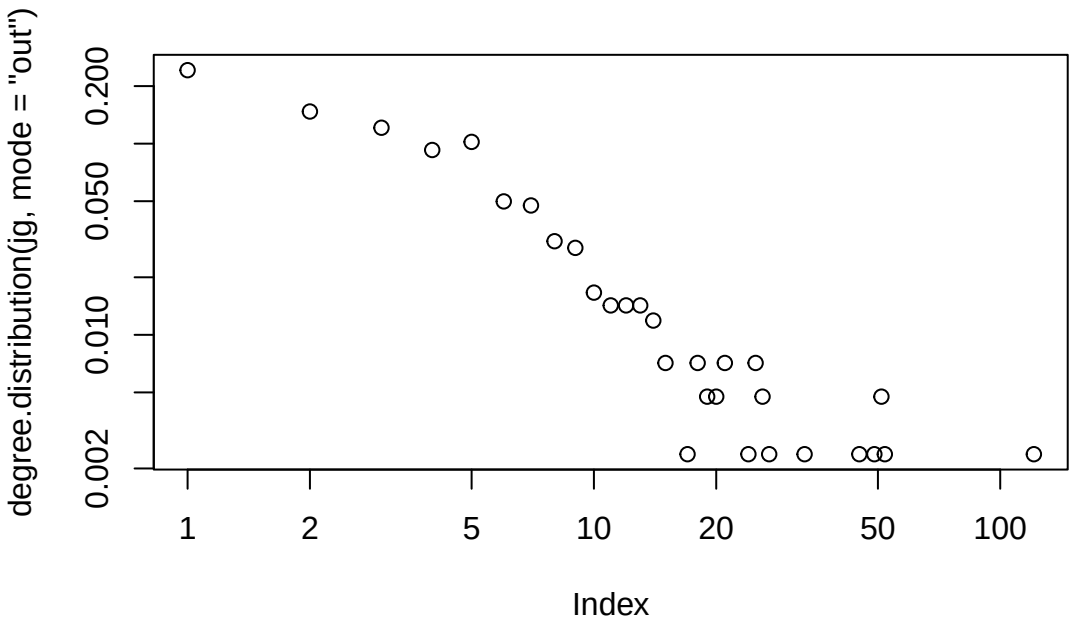


Figure 126: Larger word network with kk layout and edge-out density

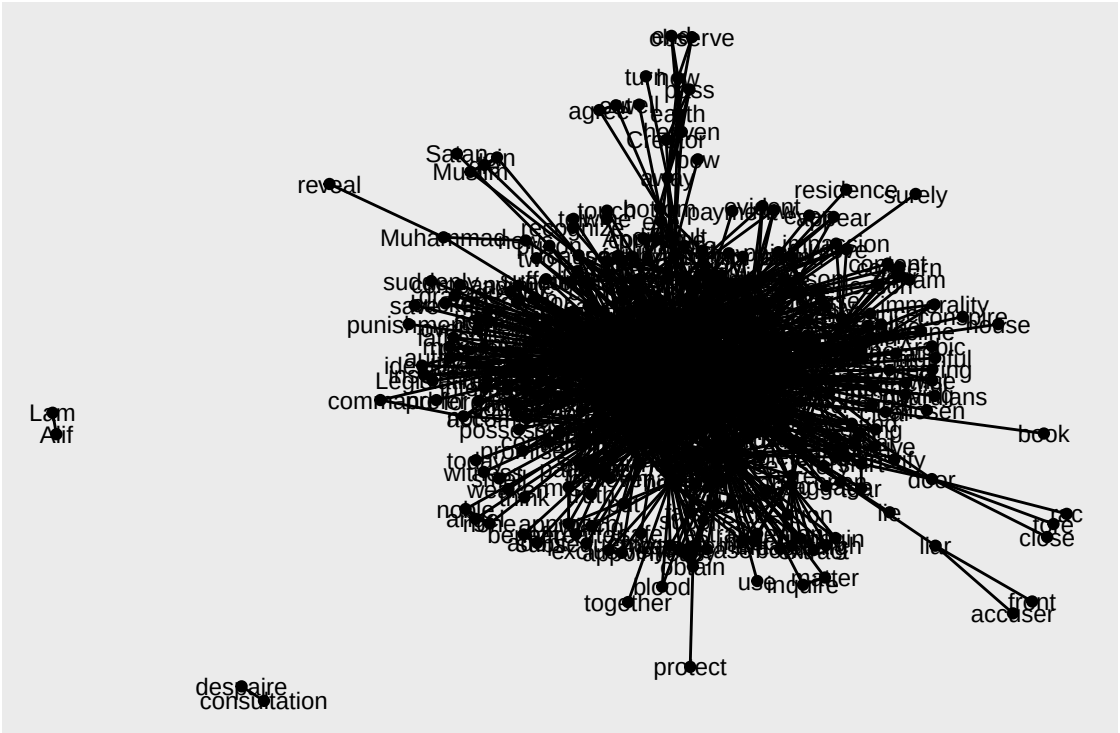


Figure 127: One big cluster and unconnected nodes

5.4.2 Taking the largest component

```
cl <- clusters(jg)
jg1 <- induced_subgraph(jg, which(cl$membership == which.max(cl$csize)))
summary(jg1)
```

Another approach is shown below.

```
jg2 <- induced_subgraph(jg,
          V(jg)[components(jg)$membership == which.max(components(jg)$csize)])
V(jg2)[1:7]
E(jg2)[1:5]
E(jg2)[6:10]
```

5.4.3 Community structure detection

```
fc <- fastgreedy.community(simplify(as.undirected(jg2)))
plot(jg2, vertex.color=vertex_attr(jg2)$cor,
      vertex.size = 2,
      vertex.label=NA,
      edge.width = NA,
      edge.color = NA,
      layout = layout_with_kk,
      mark.groups = fc,
      mark.border=NA)
```

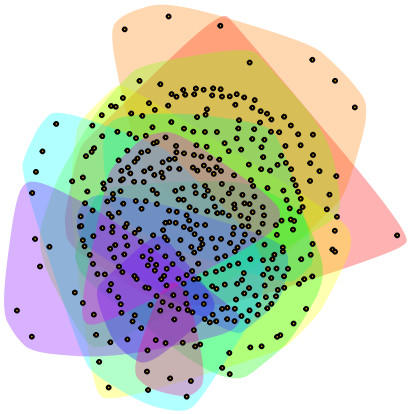


Figure 128: Communities within the large fully connected network

Plot output is in Figure 128.

The *igraph* documentation[66] lists the following for clusters and communities. Interested readers can explore the different functions following the example above.

- cluster.distribution Connected components of a graph
- cluster_edge_betweenness Community structure detection based on edge betweenness
- cluster_fast_greedy Community structure via greedy optimization of modularity

[66] https://igraph.org/r/doc/

- cluster_infomap Infomap community finding
- cluster_label_prop Finding communities based on propagating labels
- cluster_leading_eigen Community structure detecting based on the leading eigenvector of the community matrix
- cluster_louvain Finding community structure by multi-level optimization of modularity
- cluster_optimal Optimal community structure
- cluster_spinglass Finding communities in graphs based on statistical meachanics
- cluster_walktrap Community strucure via short random walks

Let us explore cluster_edge_betweenness. We can see a pleasing digital art representing the words in Surah Yusuf.

```
fc <- cluster_edge_betweenness(simplify(as.undirected(jg2)))
plot(jg2, vertex.color=vertex_attr(jg2)$cor,
      vertex.size = 2,
      vertex.label=NA,
      edge.width = NA,
      edge.color = NA,
      layout = layout_with_kk,
      mark.groups = fc,
      mark.border=NA)
```

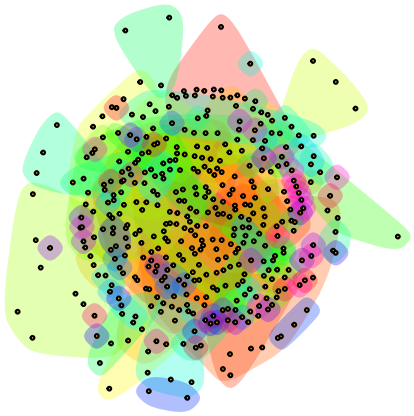


Figure 129: Another piece of art from Surah Yusuf

Plot output is in Figure 129.

5.5 Summary

Our work on Quran Analytics relies heavily on the use of network graphs. In this chapter, we introduced the tools available in R for creating, plotting, and analyzing network graphs. We have a more complete version of this tutorial.[67]

We strongly recommend our users experiment with the different parameters and functions from *ggraph* based on the sample network graphs from Surah Yusuf that we showed how to create in this chapter. It is important to have a working knowledge of network graphs to follow the coming chapters. Network graphs have applications in many other subject matters, so our readers will benefit much from this knowledge.

[67] https://rpubs.com/azmanH/696047

We will explore the *tidygraph* package and also some of the numerical analysis on network graphs like measures of centrality, in the next chapter. We have seen in the final example, concepts like "hub" (most central node) and "spread" for our Surah Yusuf word co-occurrence network that is similar to virus networks.

5.6 Further readings

Examples of *ggraph* :

- http://users.dimi.uniud.it/~massimo.franceschet/ns/syllabus/make/ggraph/ggraph.html

Examples of network analysis and visualization with R and igraph:

- https://www.kateto.net
- https://kateto.net/networks-r-igraph
- https://kateto.net/network-visualization

R graph gallery:

- https://www.r-graph-gallery.com/network.html
- https://www.data-imaginist.com/2017/ggraph-introduction-layouts/

6 Word Cooccurences of Surah Taa Haa

In Chapter 5, we discussed the network graph tools in R like *igraph* and *ggraph*. The focus was on creating the network of word co-occurrences from Surah Yusuf and then showing the many options in plotting and visualizing. In the summary section of the chapter, we mentioned another important aspect of network graphs, their characteristics, and other numerical statistics. We will refer to terms like diameter, paths and distances, connectedness, clustering, centrality, and modularity measures of the overall Quran word network in Chapter 7.

This chapter will serve as a customized tutorial on the characteristics and statistics of network graphs. We will however choose a new Surah to analyze while learning the related functions in **R**.

In the summary section of Chapter 4, we mentioned that the study opens other investigation avenues like looking into other Surahs of the Quran or analyzing different aspects of the Quran structure. So, in this chapter, we will analyze Surah Taa-Haa (Surah 20) which details the story of Prophet Musa (Moses/Mose). We will discuss centrality and other important network graph measures. We will also explore some layouts provided by *tidygraph* and *ggraph* that highlight these measures.

6.1 Data preprocessing

We repeat the same sequence of steps as in Chapter 5 except that we filter to select Surah Taa-Haa from the Saheeh International Quran *data.frame* (quran_en_sahih %>% filter(surah == 20)).

```
# For the first time model download and execute the below line too
# udmodel <- udpipe_download_model(language = "english")
# Load the model
udmodel <- udpipe_load_model(file = 'english-ewt-ud-2.5-191206.udpipe')
```

We start by annotating Surah Taa-Haa.

```
# Select the surah
Q01 <-quRan::quran_en_sahih %>% filter(surah == 20)
x <- udpipe_annotate(udmodel, x = Q01$text, doc_id = Q01$ayah_title)
x <- as.data.frame(x)
```

Again we look at how many times nouns, proper nouns, adjectives, verbs, adverbs, and numbers are used in the same verse in Surah Taa-Haa.

```
cooccur <- cooccurrence(x = subset(x, upos %in% c("NOUN", "PROPN", "VERB",
                                                  "ADJ", "ADV", "NUM")),
                        term = "lemma",
                        group = c("doc_id", "paragraph_id", "sentence_id"))
head(cooccur)
```

The result can be easily visualized using the *igraph* and *ggraph* packages. We chose the top 50 occurrences for the tutorial so the plots do not clutter.

```
wordnetwork <- head(cooccur, 50)
wordnetwork <- graph_from_data_frame(wordnetwork)
ggraph(wordnetwork, layout = "kk") +
  geom_edge_link(aes(width = cooc, edge_alpha = cooc),
                 edge_color = "deeppink") +
  geom_node_point(aes(size = igraph::degree(wordnetwork)),
                  shape = 1, color = "black") +
  geom_node_text(aes(label = name), col = "darkblue", size = 3) +
  labs(title = "Co-occurrences Within Sentence",
       subtitle = "Top 50 Nouns, Names, Adjectives, Verbs, Adverbs")
```

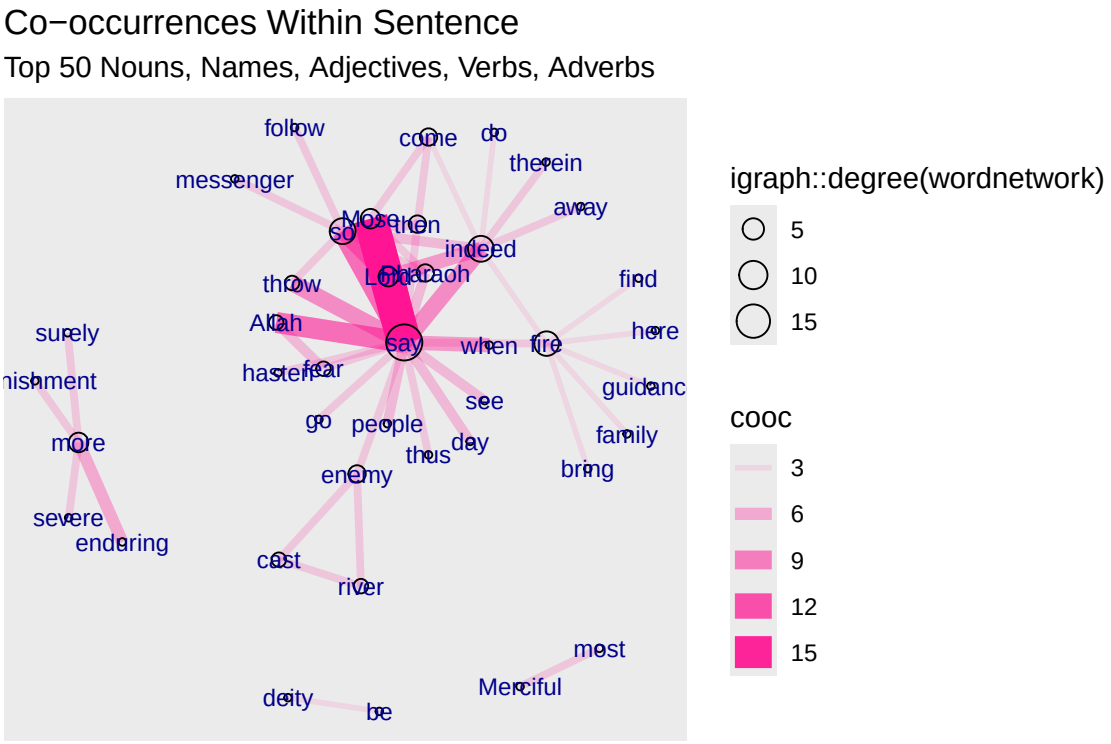


Figure 130: First plot of tutorial word network

The base wordnetwork graph is a directed network with 41 nodes and 50 edges *(DN– 41 50 –)*. The nodes have the *name* attribute. The edges have the *cooc* attribute (+ attr: name (v/c), cooc (e/n))

The story is revealed by Allah (SWT). The main characters are Mose, his brother Aaron, Pharaoh, and the magicians. So the verb "say" dominates since it is a narrated story. It is interesting to see the strong link and occurrence of "fear" with "Allah".

6.2 Network analysis and characteristics

This introductory tutorial should be useful for those new to network graphs.[68] We will be using the network graph tools frequently in our work, thus a tutorial on the network characteristics using an example from the Quran should be helpful.

We will be looking at various functions related to

- graph_measures: Graph measurements
- centrality: Calculate node and edge centrality
- group_graph: Group nodes and edges based on community structure

This set of functions provide wrappers to several graph statistic algorithms in *ìgraph*. They are intended for use inside the *tidygraph* framework and some should not be called directly. Thus we will mix and match the use of the relevant *igraph* and/or *tidygraph* functions.

We will follow the structured presentation on Network Characteristics[69] and Centrality Measures[70] for easy reference.

6.2.1 Network characteristics

Our *wordnetwork graph* is a directed network. We will also create an undirected version of the network.

```
gd <- wordnetwork
gu <- simplify(as.undirected(wordnetwork))
V(gd)[1:7]
E(gd)[1:4]
E(gd)[5:8]
V(gu)[1:7]
E(gu)[1:4]
E(gu)[5:8]
```

The *igraph* notation for directed edges uses -> (Mose ->say) and – for undirected edges (Mose –Allah).

Generally speaking, we can measure network properties at the level of nodes (centrality measures) or at the level of the network (global measures). If we are interested in the position of nodes within a network, then we are measuring something at the node level. If we want to understand the structure of the network as a whole, we are measuring something at the network level. Network analysis often combines both.

We will use the two networks, *gu* and *gd*, that we created in this section. For the early examples, we will mainly use the *igraph* package and the basic plot functions. In the later sections, we will show some of the same examples together with new ones using the *tidygraph* and *ggraph* packages.

Re-plotting of *gd* graph is in Figure 131.

[68] http://networksciencebook.com
[69] https://dshizuka.github.io/networkanalysis/04_measuring.html
[70] http://www2.unb.ca/~ddu/6634/Lecture_notes/Lecture_4_centrality_measure.pdf

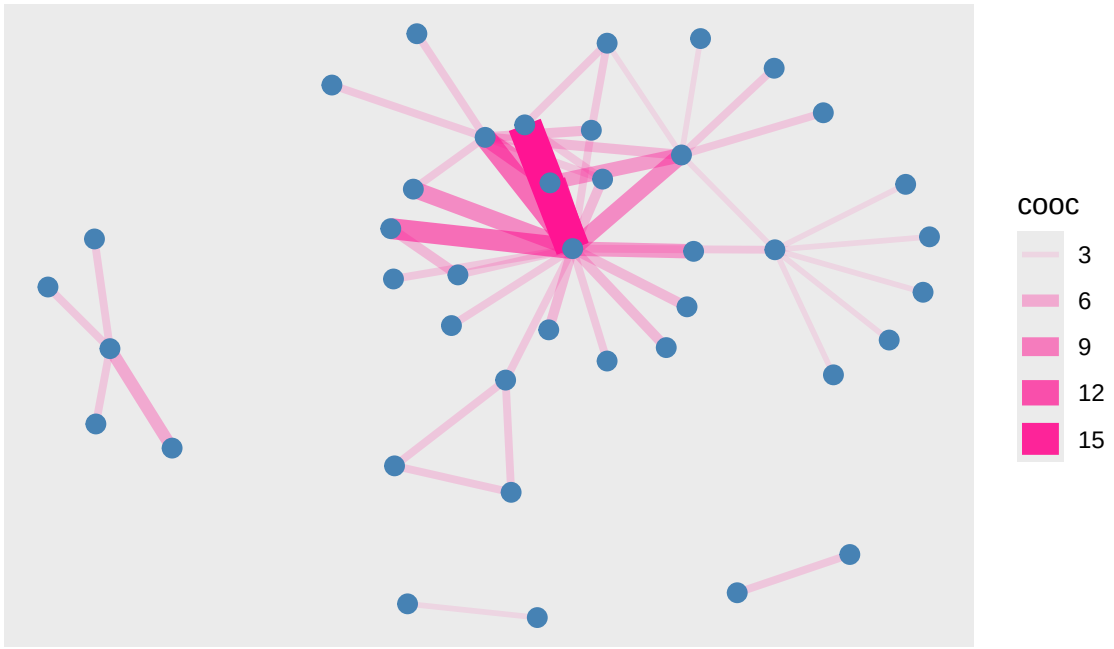


Figure 131: Second plot of tutorial word network

6.2.2 Centrality measures (node-level measures)

Centrality relates to measures of a node's position in the network. The main objective is to understand the position and/or importance of a node in the network. The individual characteristics of nodes can be described by[71]

- Degree
- Clustering
- Distance to other nodes

The *centrality* of a node reflects its influence, power, and importance. There are four different types of centrality measures.

- Degree - connectedness
- Eigenvectors - Influence, Prestige, "not what you know, but whom you know"
- Betweenness - importance as an intermediary, connector
- Closeness, Decay – ease of reaching other nodes

There are many such centrality measures. It can be difficult to go through all of the available centrality measures. We will introduce just a few examples. More examples are shown in our earlier work.[72]

- Degree centrality
- Betweenness centrality
- Closeness centrality

[71] http://web.stanford.edu/~jacksonm/Jackson-IntroConcepts.pdf
[72] https://rpubs.com/azmanH/708667

- Eigenvector centrality
- PageRank centrality

Figure 132 summarizes some of the centrality measures in a graphical format.

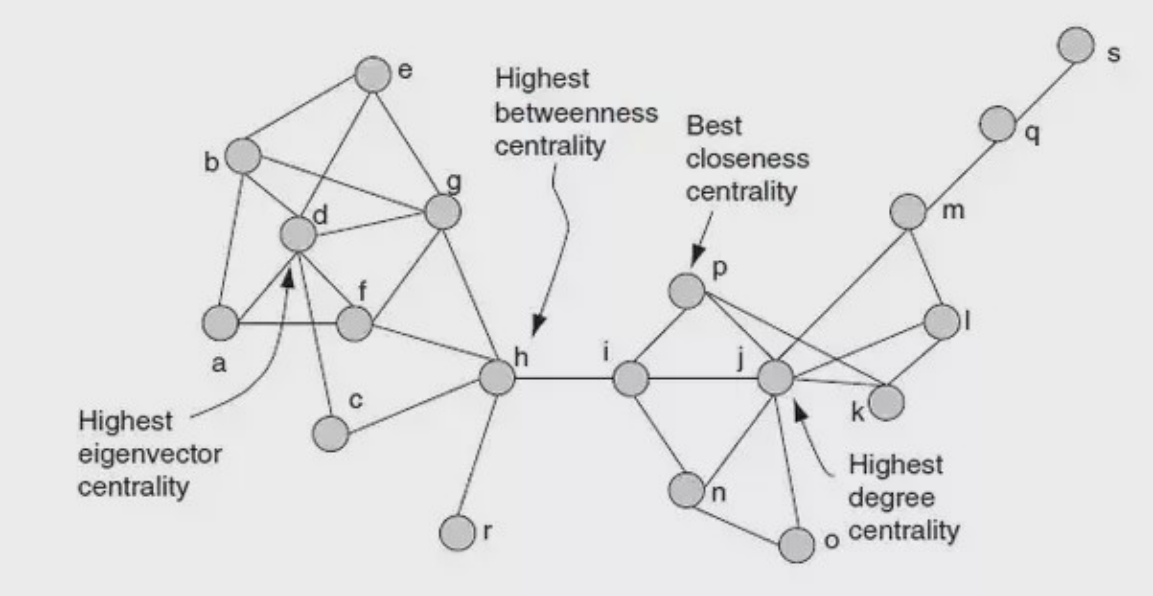


Figure 132: Centrality summary

6.2.3 Degree and strength

The most straightforward centrality measure is degree centrality. Degree centrality is simply the number of edges connected to a given node. In a social network, this might mean the number of friends an individual has. We will calculate and visualize the degree centrality by varying the node sizes proportional to degree centrality.

This is shown in Figure 133.

```
degree(gd)[1:5]
degree(gd)[6:10]
set.seed(10)
deg = igraph::degree(gd)
sort(deg, decreasing = TRUE)[1:5]
sort(deg, decreasing = TRUE)[6:10]
ggraph(gd, layout = "kk") +
  geom_edge_link(aes(width = cooc, edge_alpha = cooc),
                 edge_color = "lightseagreen") +
  geom_node_point(size = deg, color = "gold3")
```

In weighted networks, we can also use node strength, which is the sum of the weights of edges connected to the node. Let us calculate node strength and plot the node sizes as proportional to these values.

This is shown in Figure 134.

Figure 133: Node size reflects degree

```
set.seed(10)
st = graph.strength(gd)
sort(st, decreasing = TRUE)[1:5]
sort(st, decreasing = TRUE)[6:10]
ggraph(gd, layout = "kk") +
  geom_edge_link(aes(width = cooc, edge_alpha = cooc),
                 edge_color = "lightseagreen") +
  geom_node_point(size = st, color = "gold3") +
  geom_node_text(aes(filter=(st >= 3), size=st*2, label=name), repel=F)
```

Compare the relative node sizes when plotting by *degree* vs. *strength*. What differences do you notice? The top six words are the same say (18), indeed (8), so (8), fire (7), Mose (4), Lord (4).

6.2.4 Degree distribution

Degree distribution: A frequency count of the occurrence of each degree.

Let N be the number of nodes, and L be the number of edges. Average degree = 2L/N = 2(50)/41 = 2.439 for gu. The histogram of the degree is shown in Figure 135.

```
sort(degree(gu), decreasing = TRUE)[1:5]
sort(degree(gu), decreasing = TRUE)[6:10]
mean(degree(gu))
degree(gu) %>% sum()
degree.distribution(gu)[1:5]
degree.distribution(gu)[6:10]
hist(degree.distribution(gu))
```

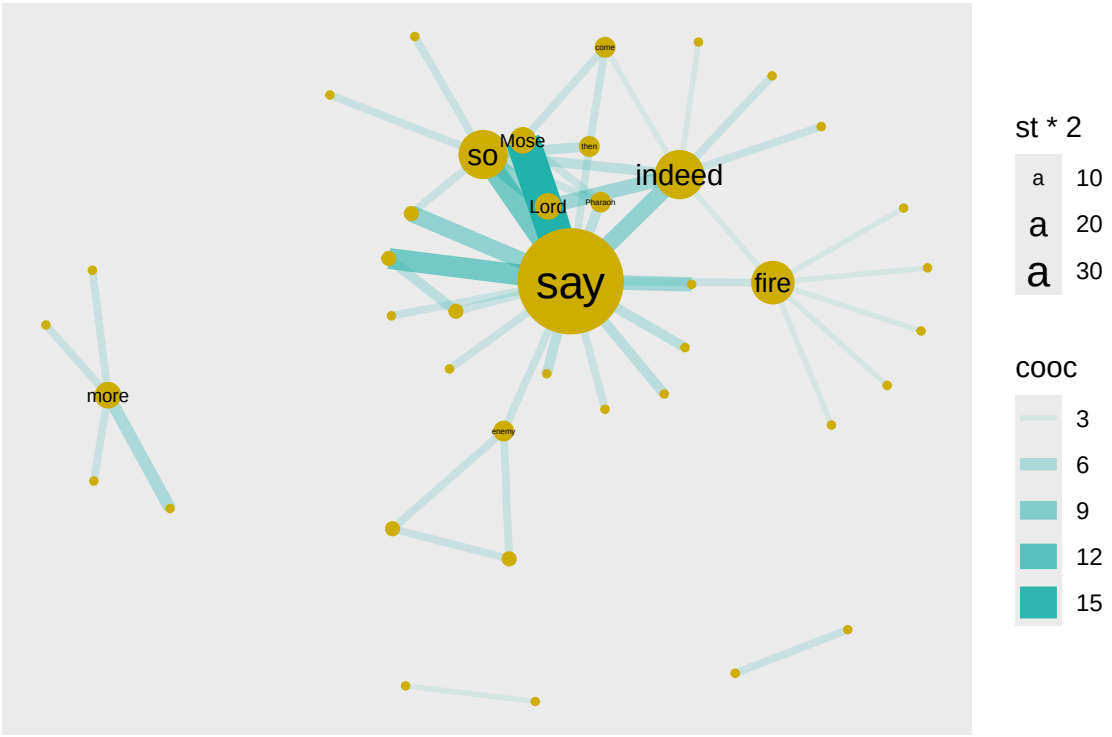


Figure 134: Node size reflects graph.strength

6.2.5 Degree and degree distribution for directed graph

- In-degree of any node i: the number of nodes ending at i.
- Out-degree of any node i: the number of nodes originating from i.
- Every loop adds one degree to each of the in-degree and out-degree of a node.
- Degree distribution: A frequency count of the occurrence of each degree
- Average degree: let N = number of nodes, and L = number of edges:
  - Avg. degree-in = Avg. degree-out = L/N = 50/41 = 1.219, for *gd*.

The plots of the histogram are in Figure 136 and Figure 137.

```
degree(gd,mode="in",loops = FALSE)[1:5]
degree(gd,mode="out",loops = FALSE)[1:5]
degree(gd,mode="in",loops = FALSE) %>% mean()
degree(gd,mode="out",loops = FALSE) %>% mean()
hist(degree.distribution(gd, mode="in"))
```

6.2.6 Why do we care about degree?

- Simplest, yet very illuminating centrality measure in a network:
  - In a social network, the ones who have connections to many others might have more influence, more access to information, or more prestige than those who have fewer connections.

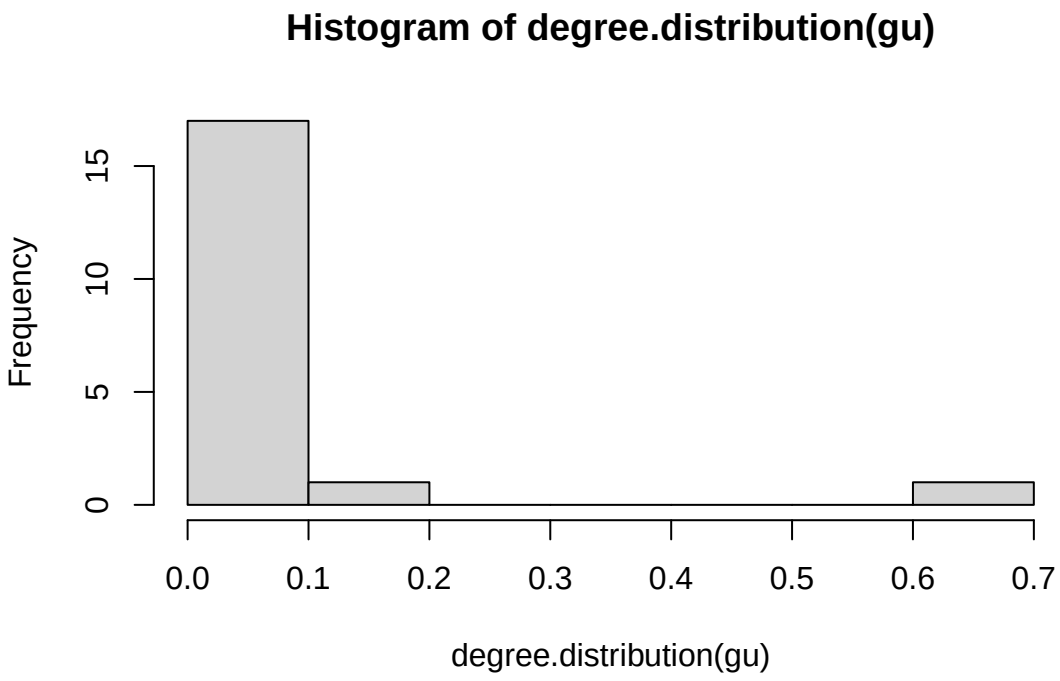


Figure 135: Degree distribution of tutorial word network

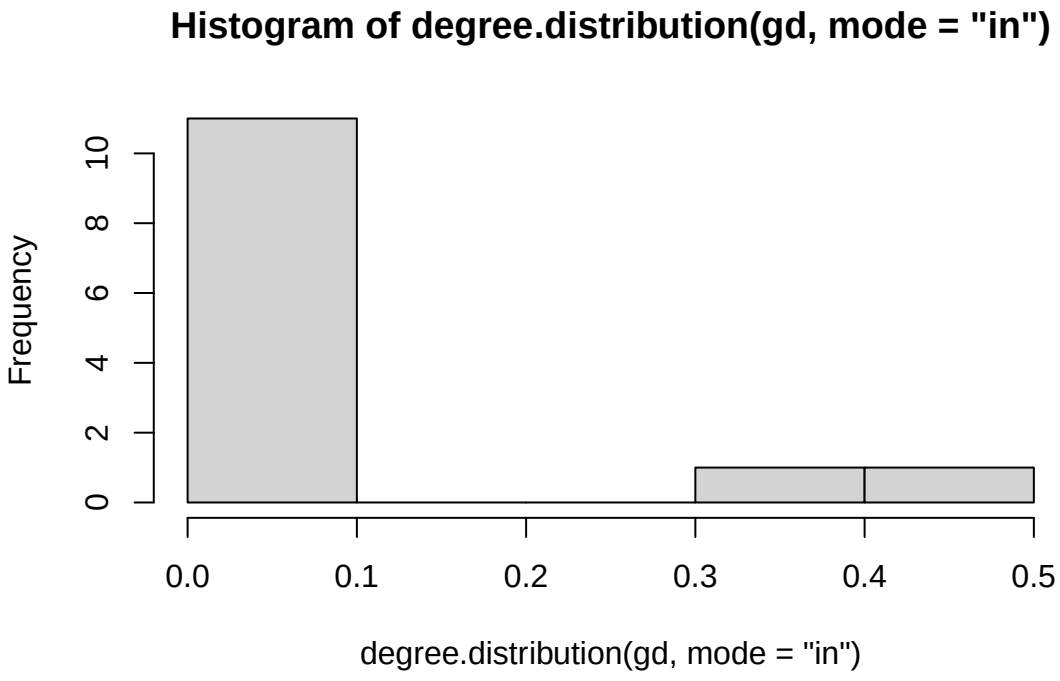


Figure 136: In degree distribution of directed word network

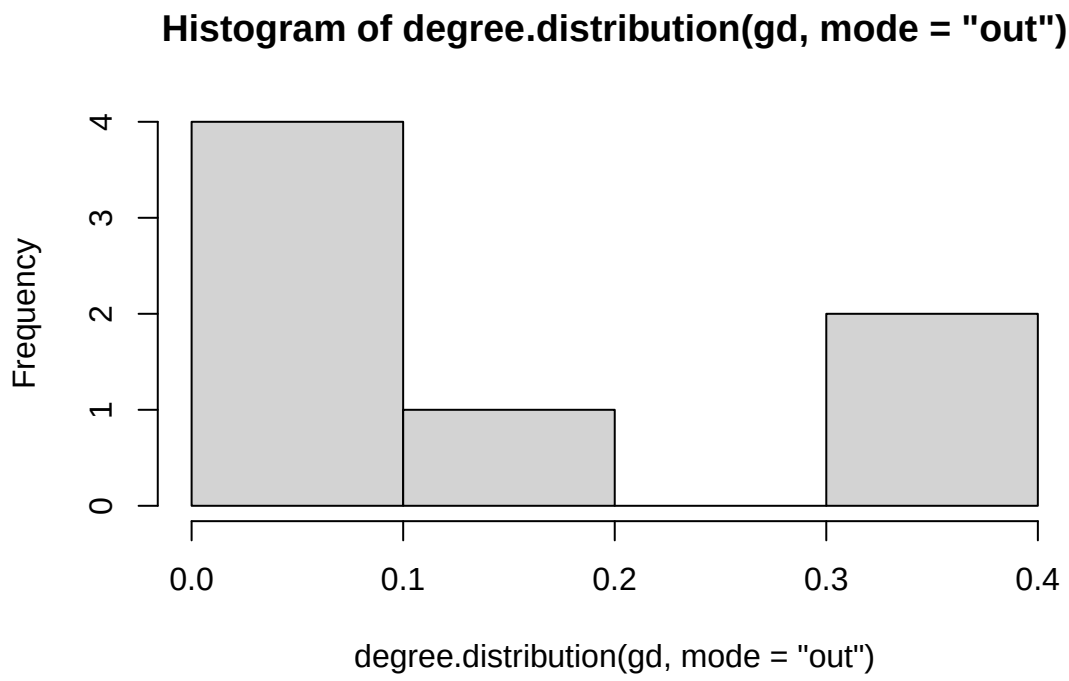


Figure 137: Out degree distribution of directed word network

- The degree is the immediate risk of a node for catching whatever is flowing through the network (such as a virus, or some information)

6.2.7 Betweenness

We now do the same for betweenness centrality. It is defined as the number of geodesic paths (shortest paths) that go through a given node. Nodes with high betweenness might be influential in a network if, for example, they capture the most amount of information flowing through the network because the information tends to flow through them.

- Betweenness centrality quantifies the number of times a node acts as a bridge along the shortest path between two other nodes.
- It was introduced as a measure for quantifying the control of a human on the communication between other humans in a social network.
- In this conception, nodes that have a high probability to occur on a randomly chosen shortest path between two randomly chosen nodes have a high betweenness.

This is shown in Figure 138.

```
betw = betweenness(gd, normalized=F)
# calculate the betweenness centrality
sort(betweenness(gu), decreasing = TRUE)[1:5]
# calculate the standardized betweenness centrality
betwS <- 2*betweenness(gu)/((vcount(gu) - 1)*(vcount(gu)-2))
sort(betwS, decreasing = TRUE)[1:5]

ggraph(gd, layout = "kk") +
  geom_edge_link(aes(width = cooc, edge_alpha = cooc),
```

```
    edge_color = "lightseagreen") +
  geom_node_point(size = betw*0.2, color = "gold3") +
  geom_node_text(aes(filter=(betw >= 5), size=betw*2, label=name), repel=F)
```

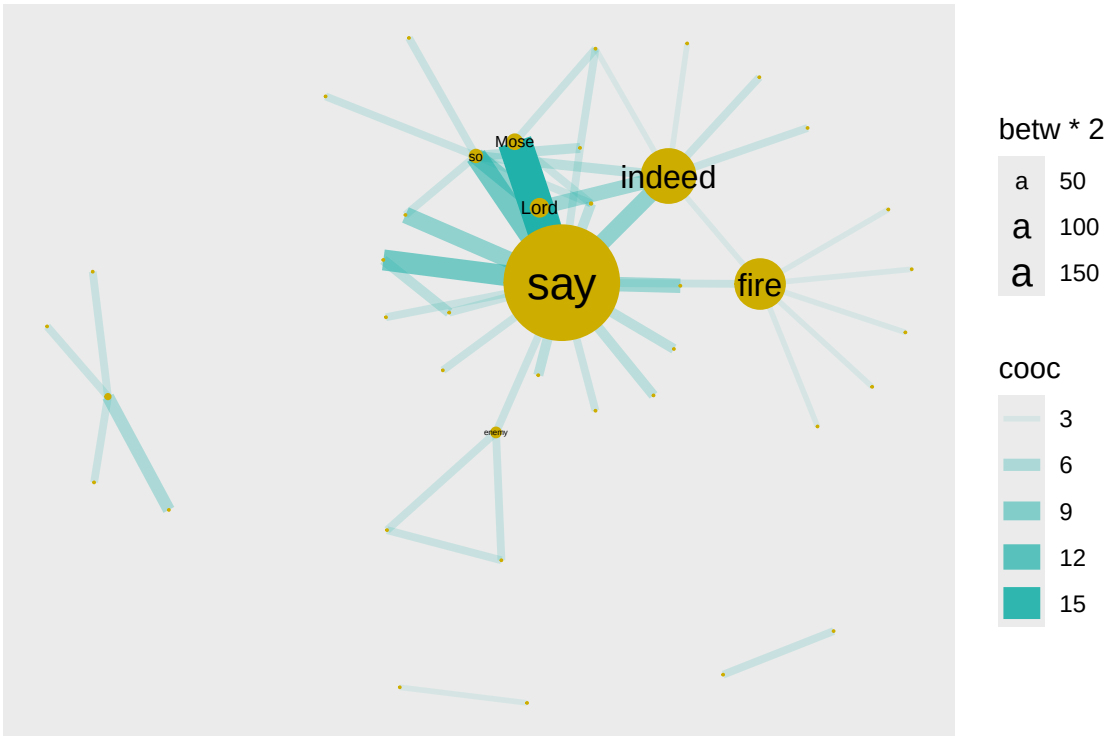


Figure 138: Node size reflects betweenness centrality

We can see that there are three nodes (words = "say", "indeed", "fire") that have qualitatively higher betweenness values than all other nodes in the network. One way to interpret this is that these are nodes that tend to act as "bridges" between different clusters of nodes in the network.

6.2.8 Degree centrality for undirected graph

- The nodes with a higher degree are more central.
- Degree is simply the number of nodes at distance one.
- Though simple, the degree is often a highly effective measure of the influence or importance of a node.
- In many social settings, people with more connections tend to have more power and are more visible.
- Group-level centralization: degree, as an individual-level centrality measure, has a distribution that can be summarized by its mean and variance as is commonly practiced in data analysis.

```
# calculate the degree centrality for business network
deg <- degree(gu, loops = FALSE)
sort(deg, decreasing = TRUE)[1:7]
# calculate the standardized degree centrality
degS <- degree(gu, loops = FALSE)/(vcount(gu) - 1)
sort(degS, decreasing = TRUE)[1:7]
```

6.2.9 Outdegree centrality and indegree prestige

- The nodes with higher out-degree are more central (choices made).

- The nodes with higher in-degree are more prestigious (choices received).

```
sort(degree(gd, mode='in'), decreasing = TRUE) %>% head(10)
sort(degree(gd, mode='out'), decreasing = TRUE) %>% head(10)
```

What does this say about the importance of these nodes? Well, that depends on the network and the questions; in particular how we might quantify 'importance' in our network. But clearly "say", "indeed" "Mose", "Allah", "Lord", "Pharaoh" are important words in the Surah. We have explained the importance of "say" in that the Surah is telling a story. "indeed" shows that the Quran is stressing the truth of the narrations.

Here is a shortlist of some commonly-used centrality measures:

- degree() - Number of edges connected to node
- graph.strength() - Sum of edge weights connected to a node (aka weighted degree)
- betweenness() - Number of geodesic paths that go through a given node
- closeness() - Number of steps required to access every other node from a given node
- eigen_centrality() - Values of the first eigenvector of the graph adjacency matrix. The values are high for nodes that are connected to many other nodes that are, in turn, connected to many others, etc.

6.2.10 Closeness centrality for undirected graph

- The farness/peripherality of a node v is defined as the sum of its distances to all other nodes
- The closeness is defined as the inverse of the farness.
- For comparison purpose, we can standardize the closeness by dividing by the maximum possible value *1/(n - 1)*
- If there is no (directed) path between node v and i then the total number of nodes is used in the formula instead of the path length.
- The more central a node is, the lower its total distance to all other nodes.
- Closeness can be regarded as a measure of how long it will take to spread information from v to all other nodes sequentially.

```
sort(closeness(gu), decreasing = TRUE) %>% head(6)
# calculate the standardized closeness centrality
closeS <- closeness(gu)*(vcount(gu) - 1)
sort(closeS, decreasing = TRUE) %>% head(6)
```

From the various plots in the earlier sections, there are some words outside the main network cluster. They are *disconnected* from the main network. Hence, there are some warning messages for disconnected graphs.

We will calculate the Eigenvector and PageRank centrality measures in the next section as we assemble a *data.frame* of node-level measures.

6.2.11 Correlation analysis among centrality measures for the *gu* network

```
# calculate the degree centrality
deg <- degree(gu, loops = FALSE)
sort(deg, decreasing = TRUE) %>% head(6) # sort the nodes in decreasing order
# calculate the standardized degree centrality
degS <- degree(gu, loops = FALSE)/(vcount(gu) - 1)
sort(degS, decreasing = TRUE) %>% head(6) # sort the nodes in decreasing order
# calculate the closeness centrality
close <- closeness(gu)
sort(close, decreasing = TRUE) %>% head(6)
# calculate the standardized closeness centrality
```

```
closeS <- closeness(gu)*(vcount(gu) - 1)
sort(closeS, decreasing = TRUE) %>% head(6)
# calculate the Betweenness centrality
betw <- betweenness(gu)
sort(betw, decreasing = TRUE) %>% head(6)
# calculate the standardized Betweenness centrality
betwS <- 2 * betweenness(gu)/((vcount(gu) - 1) * (vcount(gu)-2))
sort(betwS, decreasing = TRUE) %>% head(6)
# calculate the Eigenvector centrality
eigen <- evcent(gu)
sort(eigen[[1]], decreasing = TRUE) %>% head(6)
# calculate the PageRank centrality
page <- page.rank(gu)
sort(page[[1]], decreasing = TRUE) %>% head(6)
```

From the above parameters, we assemble a *data.frame* and plot a correlation matrix, (Figure 139).

```
dfu <- data.frame(degS, closeS, betwS, eigen[[1]], page[[1]])
# Pearson correlation matrix
Pearson_correlation_matrix <- cor(dfu)
# Spearman correlation matrix
Spearman_correlation_matrix <- cor(dfu, method = "spearman")
# Kendall correlation matrix
cor(dfu, method = "kendall")
# Basic Scatterplot Matrix
pairs(~deg + close + betw + eigen[[1]] + page[[1]],
      data=dfu)
```

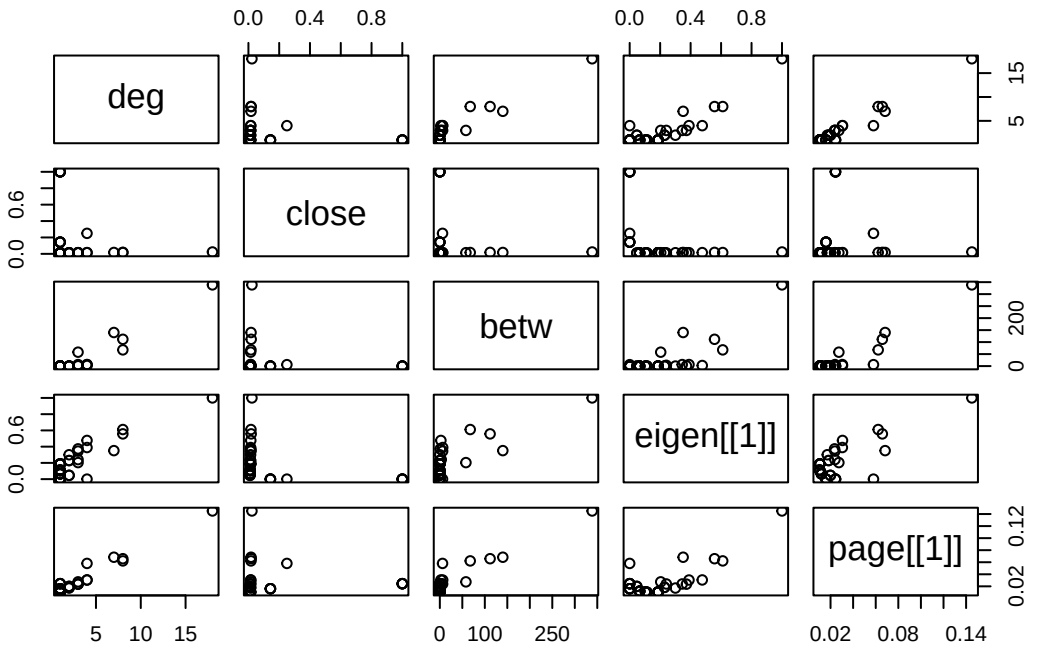


Figure 139: Simple Scatterplot Matrix

6.2.12 Assembling a dataset of node-level measures for gd network

```
#assemble dataset
names = V(gd)$name
deg = degree(gd)
st = graph.strength(gd)
betw = betweenness(gd, normalized=F)
eigen <- evcent(gd)
page <- page.rank(gd)
dfd = data.frame(node.name=names, degree=deg, strength=st,
                 betweenness=betw,
                 eigen = eigen[[1]], page = page[[1]])
head(dfd)
# plot the relationship between degree and strength
dfd %>% ggplot(aes(x = strength, y = degree)) + geom_point() +
        geom_text(label = rownames(dfd),
                  size = 3, color = "darkblue",
                  nudge_x = 0.25, nudge_y = 0.25,
                  check_overlap = T) +
        labs(title = "Word Co-occurrences Network")
```

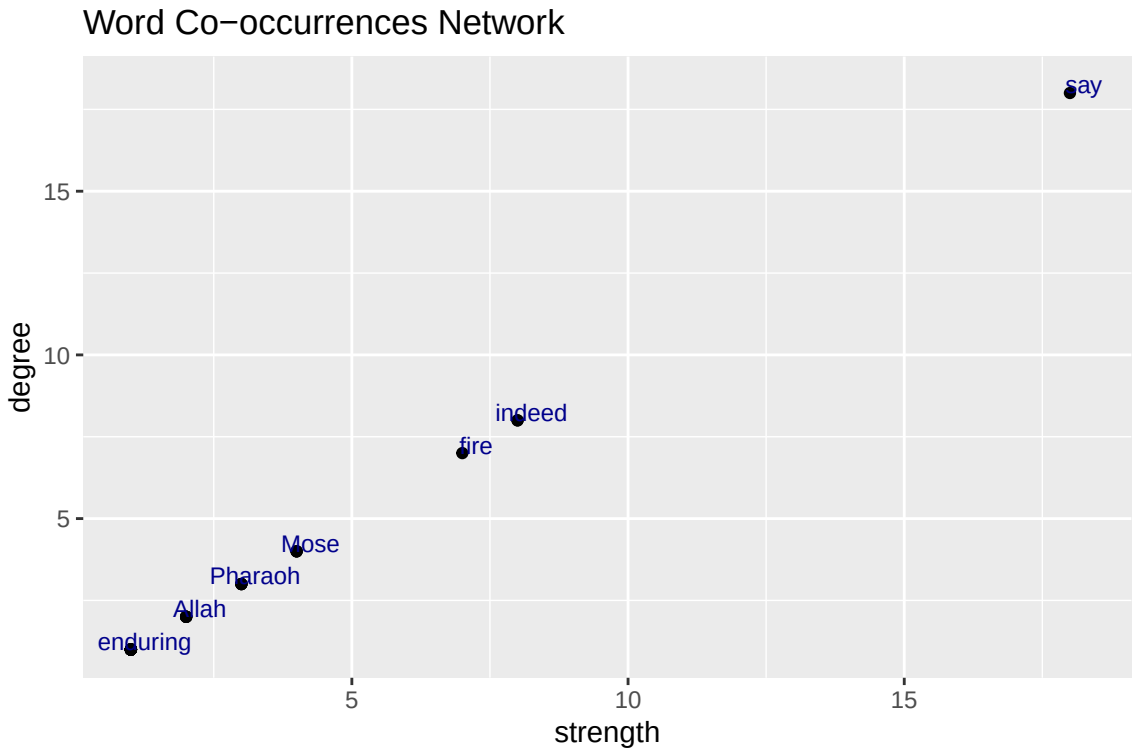


Figure 140: Relationship degree and strength

The straight line in Figure 140 obviously shows that these are correlated since strength is simply the weighted version of degree.

How about the relationship between betweenness and strength?

These are not well correlated, since they describe something different (as shown in Figure 141, points are not in a straight line). Again the common words “say” and “indeed” have a dominant role in Surah Taa-Haa

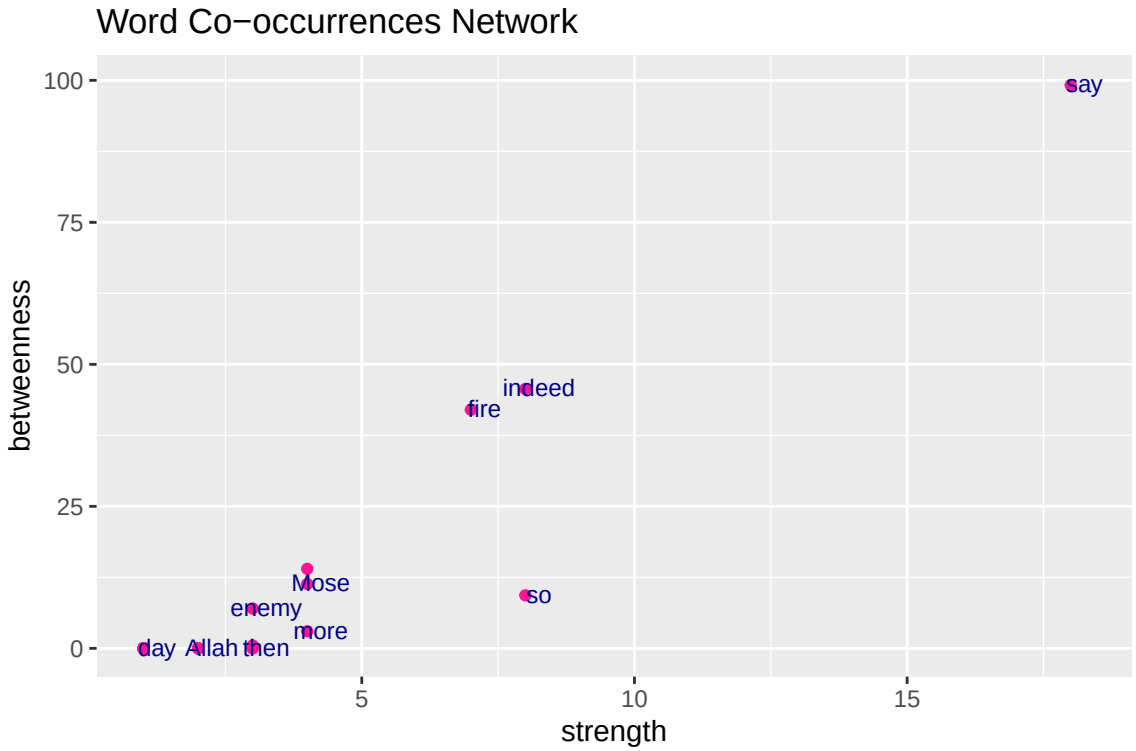


Figure 141: Relationship betweenness and strength

that narrates the true story of Prophet Mose. The common adverb "so" is often used for emphasis[73] to stress some facts and lessons of the story.

6.3 Network-level measures

6.3.1 Size and density

We start by getting some basic information for the network, such as the number of nodes and edges. There are a couple of functions to help us extract this information without having to look it up in the "object summary" (e.g., summary(gd)). Using these functions, we can store the information as separate objects, e.g., n for number of nodes and m for number of edges.

For *gd* the number of nodes n is 41 and the number of edges m is 50. For *gu* the number of nodes n is 41 and the number of edges m is 50.

The definition of network density is:

> density = [# edges that exist] / [# edges that are possible]

In an undirected network with no loops, the number of edges that are possible is exactly the number of dyads that exist in the network. In turn, the number of dyads is *n(n-1)/2* where n = number of nodes. With this information, we can calculate the density with the following:

> dyads directed = *n(n-1)* = 41(41-1) = 1640

[73] https://www.macmillandictionary.com/dictionary/british/so_1

dyads undirected = $n(n\text{-}1)/2$ = 41(41-1)/2 = 820

density = m/dyads

There is a pre-packaged function for calculating density, edge_density():

6.3.2 Components

For 'fully connected' networks, we can follow edges from any given node to all other nodes in the network. Networks can also be composed of multiple components that are not connected to each other, as is obvious from the plots of our sample word network *gd*. We can get this information with a simple function.

```
components(gd)$membership[1:7]
components(gd)$csize
components(gd)$no
plot(gd)
```

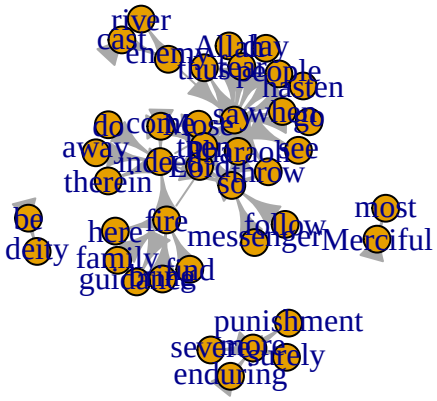


Figure 142: Simple plot showing network components

The output shows the node membership, component sizes, and number of components. The numbers for *no* (number of components, 4) and *csize* (size for each component) can be confirmed from Figure 142.

6.3.3 Degree distributions

Degree distribution, the statistical distribution of node degrees in a network, is a common and often powerful way to describe a network. We can simply look at the degree distribution as a histogram of degrees. (See the plots in Figure 143 and Figure 144).

However, if we want to compare the degree distributions of different networks, it might be more useful to plot the probability densities of each degree: i.e., what proportion of nodes has degree = 1, degree = 2, etc. We can do this by using the function *degree.distribution()*. The output of the plot is in Figure 145 and Figure 146.

Degree and degree distribution play an important role in understanding networks. A higher density changes the component structure of a network and impacts the diffusion and learning properties. The degree also is an individual node's characteristic and reflects its position.

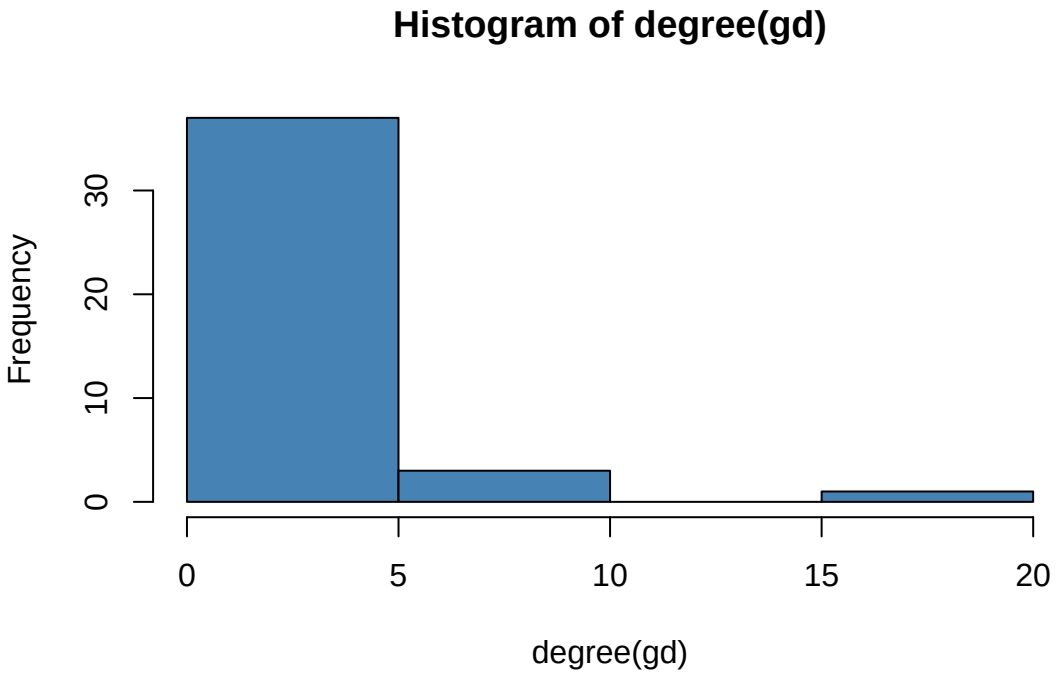


Figure 143: Histogram of gd degrees

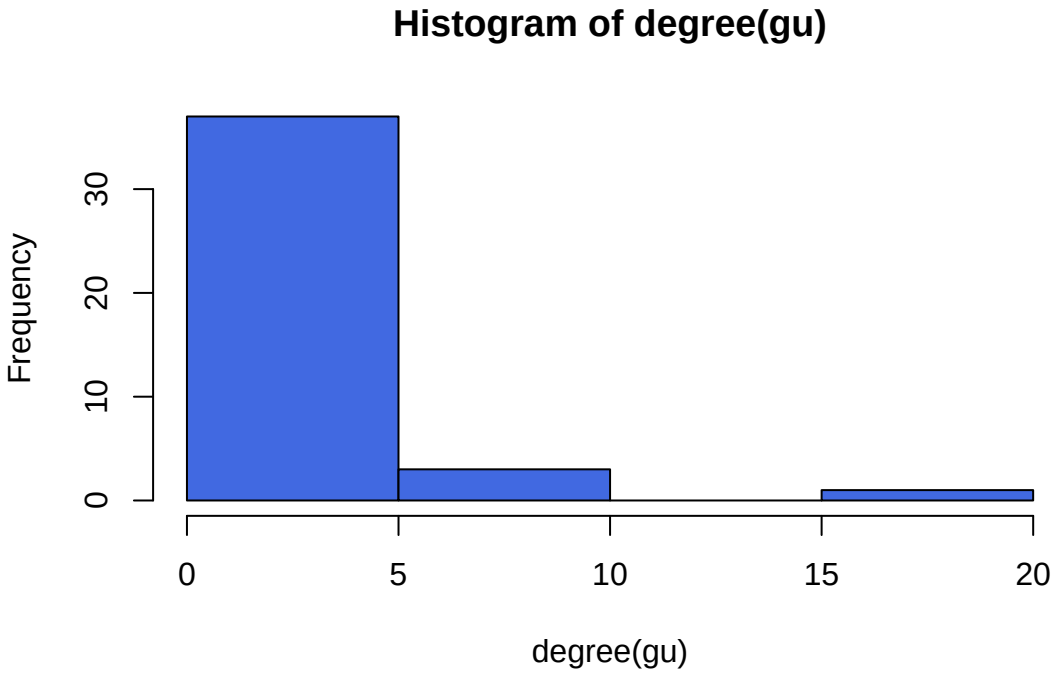


Figure 144: Histogram of gu degrees

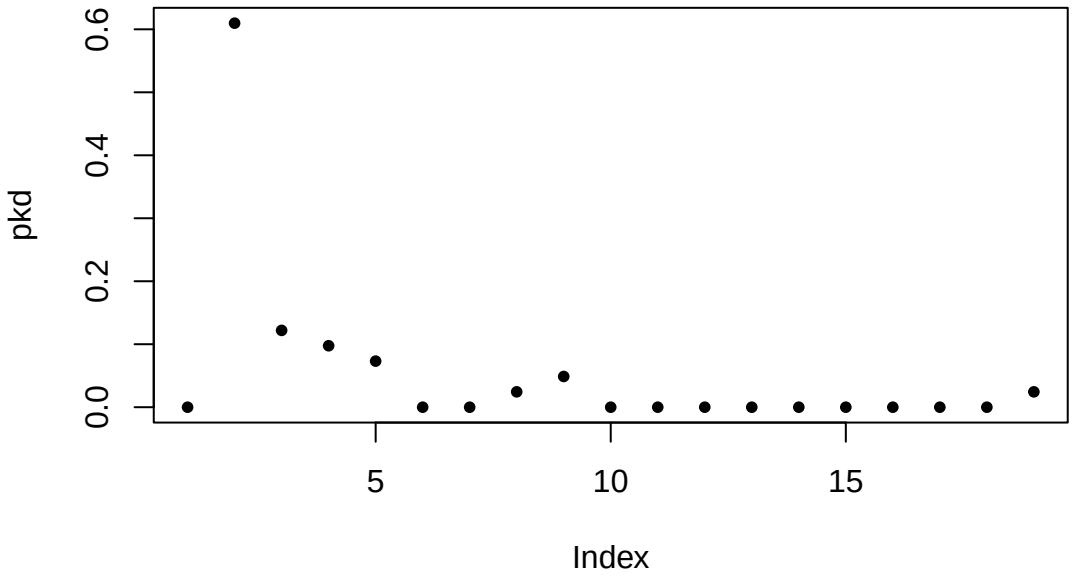


Figure 145: Degree distribution of gd

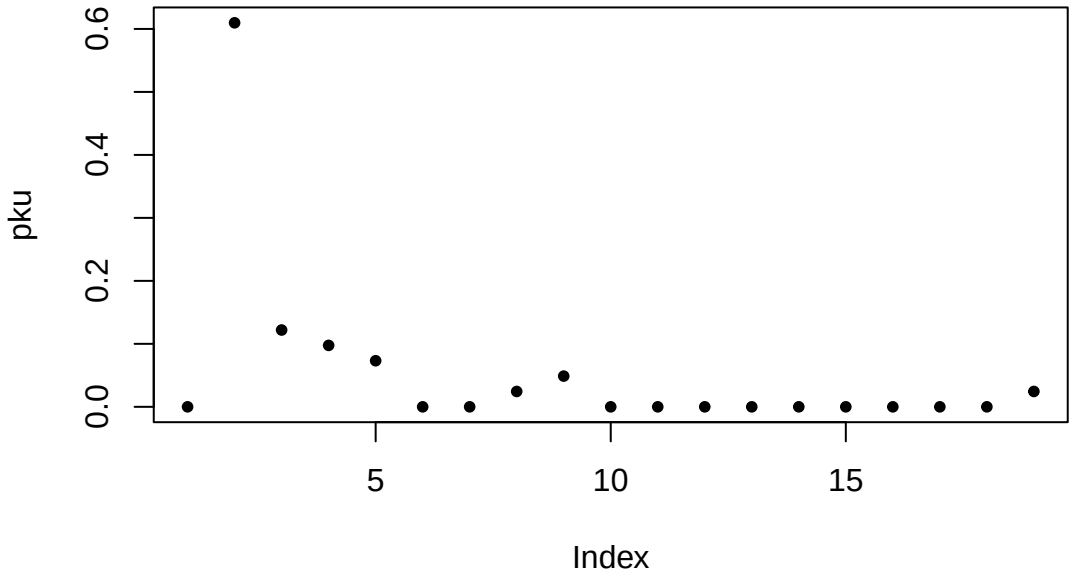


Figure 146: Degree distribution of gu

6.3.4 Average path length and diameter

The network "path" is typically a short form for "geodesic path" or "shortest path" — the fewest number of edges to get from one node to another.

- The average path length can be considered as the average "degrees of separation" between all pairs of nodes in the network.
- The diameter (maximum path length) is the maximum degree of separation that exists in the network.

We can calculate path lengths with or without the edge weights (if using edge weights, we simply count up the weights as we go along the path). The *igraph* package includes a convenient function for finding the shortest paths between every dyad in a network. Make sure algorithm is set as "unweighted".

- pathd = distances(gd, algorithm="unweighted")
- pathu = distances(gu, algorithm="unweighted")

This matrix is usually large, so we will not display the output. It gives us the geodesic path length between each pair of nodes in the network. We can describe the network using some characteristics of the paths that exist in that network. This matrix contains a bunch of cells that are "Inf" (i.e., infinity). This is because the network is not connected, and we cannot calculate path lengths between nodes in different components.

How should we measure the average path length and diameter of a network with multiple components? There are two common solutions. The first is to ignore pairs of nodes that are in different components and only measure the average lengths of the paths that exist. This solution does not make sense for the diameter since the diameter of an unconnected network should be infinity. The second solution is to measure each component separately. We will do each of these in turn.

Option 1: To calculate the average path length while ignoring pairs of nodes that are in different components, we can first replace the "Inf" with "NA" in the path length matrix. Next, we want just the "upper triangle" or "lower triangle" of this matrix, which lists all the geodesic paths without duplicates.

```
pathd = distances(gd, algorithm="unweighted")
pathu = distances(gu, algorithm="unweighted")
pathd[pathd=="Inf"]=NA
mean(pathd[upper.tri(pathd)], na.rm=T)
pathu[pathu=="Inf"]=NA
mean(pathu[upper.tri(pathu)], na.rm=T)
```

This is what the function mean_distances() does for unconnected networks because we will get the same value:

```
mean_distance(gd)
mean_distance(gu)
```

Option 2: To calculate the average path lengths and diameter separately for each component, we will first 'decompose' the network into a list that contains each component as separate graph objects. We can then use the lapply() function to calculate separate path length matrices, and sapply() function to calculate the mean and max for each matrix.

```
comps = decompose(gd)
#make list object with two path length matrices
path.list = lapply(comps, function(x) distances(x, algorithm="unweighted"))
#average path length of each component
avg.paths=sapply(path.list, mean)
#diameter of each component
diams=sapply(path.list, max)
avg.paths
diams
```

6.3.5 Path distance distribution

Path distribution: A frequency count of the occurrence of each path distance.

```
average.path.length(gu)
path.length.hist(gu)
```

- $res is the histogram of distances,
- $unconnected is the number of pairs for which the first node is not reachable from the second.

6.3.6 Path distance distribution for directed graph

```
average.path.length(gd)
path.length.hist (gd)
```

6.3.7 Why do we care about path?

- Path means connectivity.
- Path captures the indirect interactions in a network, and individual nodes benefit (or suffer) from indirect relationships because friends might provide access to favors from their friends or a virus might spread through the links of a network.
- Path is closely related to the small-world phenomenon.
- Path is related to many other centrality measures.

6.3.8 Clustering coefficient (Transitivity) distribution

There are two formal definitions of the Clustering Coefficient (or Transitivity): "global clustering coefficient" and "local clustering coefficient". They are slightly different, but both deal with the probability of two nodes that are connected to a common node being connected themselves (e.g., the probability of two of your friends knowing each other).

- Global Clustering Coefficient = "ratio of triangles to connected triples"
- Local Clustering Coefficient = for each node, the proportion of their neighbors that are connected to each other
- Average Local Clustering Coefficient: If Ci is the proportion of two nodes connected to node i that are also connected to each other (i.e., the Local Clustering Coefficient), then Average Local Clustering Coefficient = $sum(Ci)/n$

```
# global clustering: the ratio of the triangles and
# the connected triples in the graph
g.cluster = transitivity(gd, "global")
# local clustering
l.cluster = transitivity(gd, "local")
# average clustering
av.l.cluster = transitivity(gd, "localaverage")
g.cluster
l.cluster[1:6]
av.l.cluster
# undirected
g.cluster = transitivity(gu, "global")
l.cluster = transitivity(gu, "local")
```

```
av.l.cluster = transitivity(gu, "localaverage")
g.cluster
l.cluster[1:6]
av.l.cluster
```

6.3.9 Why do we care about clustering coefficient?

- A clustering coefficient is a measure of the degree to which nodes in a graph tend to cluster together. In most real-world networks, and in particular social networks, nodes tend to create tightly knit groups characterized by a relatively high density of links; this likelihood tends to be greater than the average probability of a link randomly established between two nodes.
- Nodes with higher degrees have a lower local clustering coefficient on average.
- Local clustering can be used as a probe for the existence of so-called structural holes in a network, which are missing links between neighbors of a node.
- Structural holes can be bad when we are interested in the efficient spread of information or other traffic around a network because they reduce the number of alternative routes information can take through the network.
- Structural holes can be a good thing for the central node whose friends lack connections because they give power over information flow between those friends.
- The local clustering coefficient measures how influential a node is in this sense, taking lower values the more structural holes there are in the network around it.
- Local clustering can be regarded as a type of centrality measure, albeit one that takes small values for powerful individuals rather than large ones.

6.4 Community structure and assortment

Networks exhibit community structure, the presence of discrete clusters of nodes that are densely connected, which themselves are only loosely connected to other clusters. These may be clusters of individuals that form social groups. How do we detect the presence of such clusters or communities, and how can we quantify the degree of community structure?

6.4.1 Modularity and community detection

Modularity-based methods of community detection are not fool-proof. There is no perfect approach to community detection. There are several functions available for community detection in *igraph* and other packages.

- edge.betweenness.community()
  - One of the first in the class of "modularity optimization" algorithms. It is a "divisive" method. Cut the edge with the highest edge betweenness, and recalculate. Eventually, you end up cutting the network into different groups.
- fastgreedy.community()
  - Hierarchical agglomerative method that is designed to run well in large networks. Creates "multi-graphs" where you lump groups of nodes together in the process of agglomeration to save time on sparse graphs.
- walktrap.community()
  - Uses random walks to calculate distances, and then use agglomerative method to optimize modularity
- spinglass.community()

- This method uses the analogy of the lowest-energy state of a collection of magnets (a so-called spin-glass model).

- leading.eigenvector.community()
  - This is a "spectral partitioning" method. You first define a 'modularity matrix', which sums to 0 when there is no community structure. The leading eigenvector of this matrix ends up being useful as a community membership vector.

- cluster_louvain()
  - The "Louvain" method, so-called because it was created by a group of researchers at Louvain University in Belgium.

6.4.2 Modularity and community detection: a simple example

Our undirected word co-occurrence network *gu* appears to have a clear community structure from the earlier plots (see Figure 147).

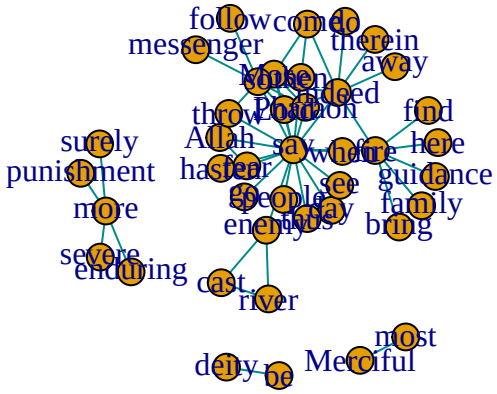


Figure 147: Layout plot of gu

Because the community division in this example is clear, we can choose any of the community detection methods as described in the previous section and we are likely to come up with the same answer.

```
eb = edge.betweenness.community(gu)
length(eb) # number of communities
modularity(eb) # modularity
```

The resulting object is a 'communities object', which includes a few pieces of information - the number of communities (groups), the modularity value based on the node assignments, and the membership of nodes to each community. We can call each of these values separately.

We can also use this 'communities object' to show the community structure (see Figure 148).

We repeat with the Louvain method (see Figure 149):

Figures 148 and 149 show that the two different methods yield different results; one with 7 and the other 8 communities (groups).

We can customize the plot. We use the *RColorBrewer* package to assign colors (see Figure 150).

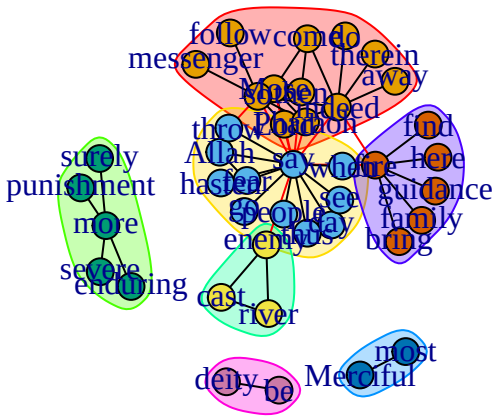


Figure 148: Community structure using edge.betweenness.community()

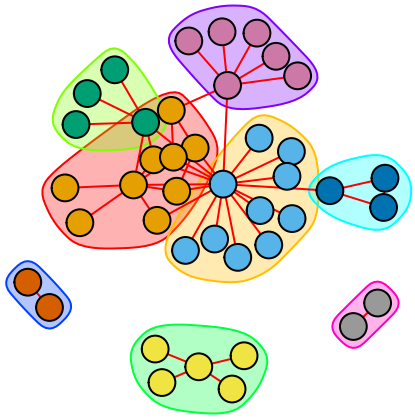


Figure 149: Community structure using the Louvain method

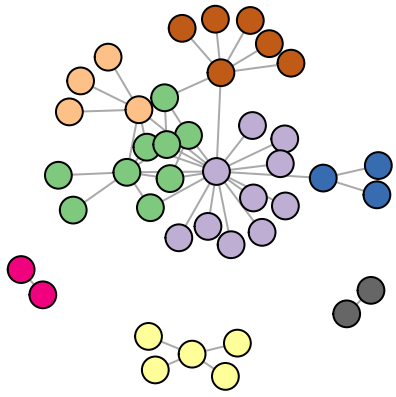


Figure 150: Using the Louvain method with some color adjustments

6.4.3 Another example of clustering

The goal of clustering (also referred to as "community detection") is to find cohesive subgroups within a network. We have mentioned earlier that there are various algorithms for graph clustering in *igraph*. It is important to note that there is no real theoretical basis for what constitutes a cluster in a network except for the vague "internally dense" versus "externally sparse" argument. As such, there is no clear argument for or against certain algorithms/methods.

No matter which algorithm is chosen, the workflow is always the same.

```
clu <- cluster_louvain(gu)
mem <- membership(clu) # membership vector
head(mem)
com <- communities(clu) # communities as list
com[[1]]
```

An example using selected graph clustering algorithms for our *gu* network is shown below.

```
scores <- c(infomap = modularity(gu,membership(imc)),
            eigen = modularity(gu,membership(lec)),
            louvain = modularity(gu,membership(loc)),
            walk = modularity(gu,membership(wtc)))
scores
```

For the *gu* network, the modularity score is around 0.5 despite the different functions. In general, though, it is advisable to use cluster_louvain() since it has the best speed/performance trade-off.

6.4.4 Assortment (homophily)

One major pattern common to many social networks (and other types of networks) is homophily or assortment — the tendency for nodes that share a trait to be connected. The assortment coefficient is a commonly used measure of homophily. It is similar to the modularity index used in community detection, but the assortativity coefficient is used when we know a priori the 'type' or 'value' of nodes. For example, we can use the assortment coefficient to examine whether discrete node types (e.g., gender, ethnicity, etc.) are more or less connected to each other. Assortment coefficient can also be used with "scalar attributes" (i.e. continuously varying traits).

Assortment coefficient

There are at least two easy ways to calculate the assortment coefficient. In the *igraph* package, there is a function for assortativity(). One benefit to this function is that it can calculate assortment in directed or undirected networks. However, the major downside is that it cannot handle weighted networks.

Assortment: a simple example with igraph

We use the same example network in this tutorial to demonstrate how to calculate assortativity, and to compare the difference between modularity and assortativity.

We start by assigning each node a value and let it vary in size (see Figure 151).

```
set.seed(3)
l = layout_with_kk(gu)
# assign sizes to nodes using two normal distributions
# with different means
V(gu)$size = 2*degree(gu)
plot(gu, layout=l, edge.color="black", repel=T)
```

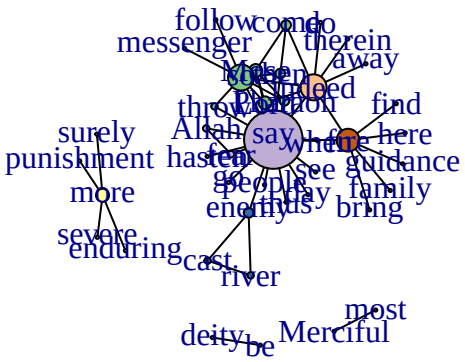


Figure 151: Node size reflecting its degree

```
assortativity(gu, V(gu)$size, directed=F)
```

This network exhibits negative (low) levels of assortativity by node size.

We can also convert the size variable into a binary (i.e., discrete) trait and calculate the assortment coefficient (see Figure 152).

```
V(gu)$size.discrete = (V(gu)$size > 5) + 0
# shortcut to make the values = 1 if large individual
# and 0 if small individual, with cutoff at size = 5
plot(gu, layout=l, edge.color="black", repel=T)
```

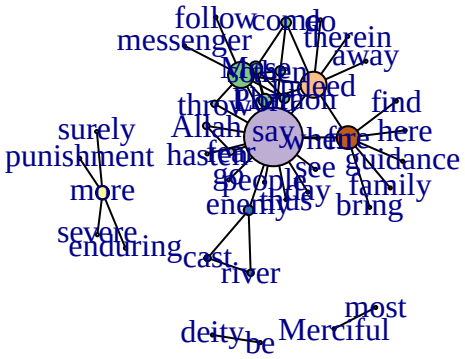


Figure 152: Node size discrete

```
assortativity(gu, V(gu)$size.discrete, directed=F)
```

As a comparison, we create a node attribute that varies randomly across all nodes in the network and then measure the assortativity coefficient based on this trait. We will plot the figure with square nodes, just to make it clear that we are plotting a different trait(see Figure 153).

```
set.seed(3)
# create a node trait that varies randomly for all nodes
V(gu)$random = rnorm(41, mean=20, sd=5)
assortativity(gu, V(gu)$random, directed=F)
plot(gu, layout=l, edge.color="black", vertex.size=V(gu)$random,
     vertex.shape="square")
```

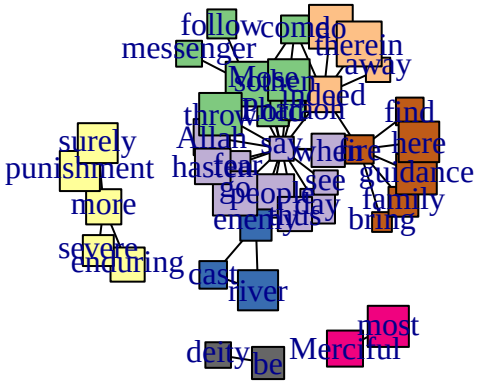


Figure 153: Using different node shape

We can see that there is little assortment based on this trait.

Just to be extra clear, this network still exhibits a community structure, but the trait we are measuring does not exhibit assortativity.

6.5 Analyzing using *tidygraph*

Earlier we created the *wordnetwork* using *igraph*. In this section, we will show similar and different examples using *tidygraph*.

Two functions of the *tidygraph* package can be used to create network objects, they are:

- tbl_graph() creates a network object from nodes and edges data.
- as_tbl_graph() converts network data and objects to a tbl_graph network.

A central aspect of *tidygraph* is to directly manipulate node and edge data from the tbl_graph object by activating nodes or edges. When we first create a tbl_graph object, the nodes will be activated. We can then directly calculate node or edge measures, like centrality, using *tidyverse* functions.

```
library(tidygraph)
gt <- as_tbl_graph(wordnetwork)
gt
```

Notice how the *igraph wordnetwork* is converted into two separate tibbles, Node Data and Edge Data. But both *wordnetwork* and *gt* are of the same *igraph class*.

6.5.1 Direct ggraph integration

gt can directly be used with our preferred *ggraph* package for visualizing networks (see the plot output in Figure 154).

```
ggraph(gt, layout = 'fr', weights = cooc) +
  geom_edge_link() +
  geom_node_point()
```

Now it is much easier to experiment with modifications to the node and edge parameters affecting layouts as it is not necessary to modify the underlying graph but only the plotting code (output in Figure 155).

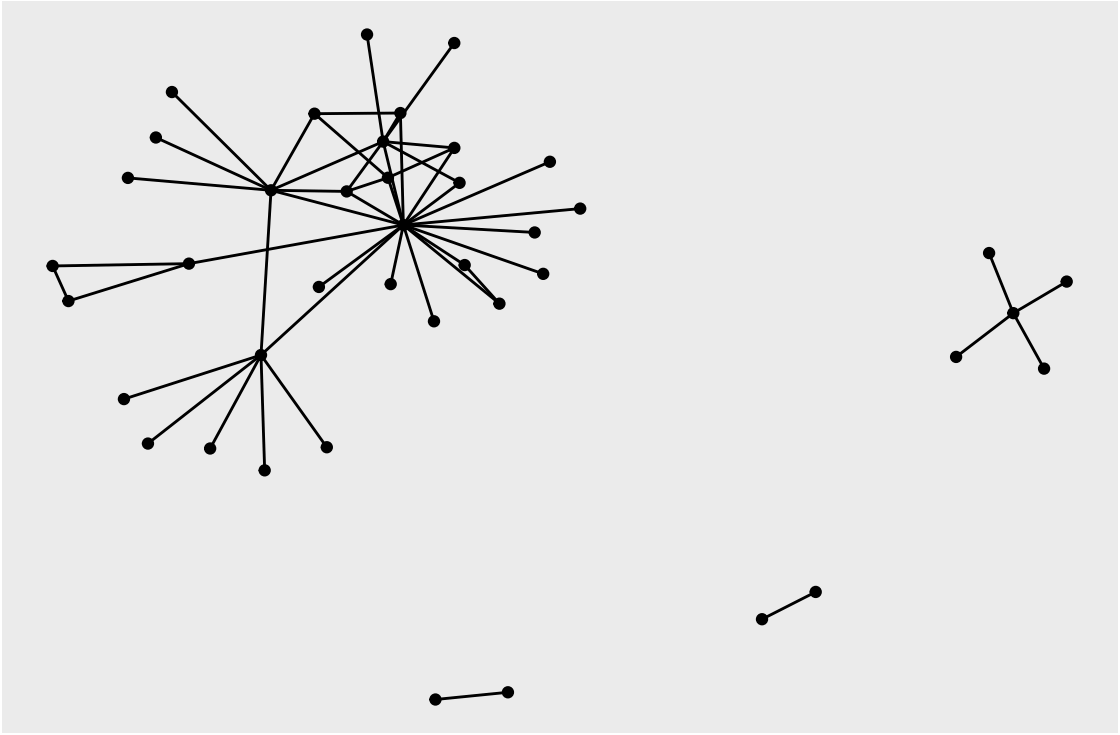


Figure 154: Simple first plot example using tidygraph and ggraph

```
ggraph(gt, layout = 'fr', weights = log(cooc)) +
  geom_edge_link(color = "cyan4") +
  geom_node_point(color = "gold4", size = 3)
```

6.5.2 Use selected measures from *tidygraph* and plot

We show below how to collate some of the node related measures. Some of the measures we have shown earlier using *igraph*. Now it appears in a tidy *data.frame* or *tibble* format.

```
node_measures <- gt %>%
     activate(nodes) %>%
     mutate(
         degree_in = centrality_degree(mode = "in"),
         degree_out = centrality_degree(mode = "out"),
         degree = degree_in + degree_out,
         betweenness = centrality_betweenness(),
         closeness = centrality_closeness(normalized = T),
         pg_rank = centrality_pagerank(),
         eigen = centrality_eigen(),
         br_score = node_bridging_score(),
         coreness = node_coreness()
     ) %>% as_tibble()
node_measures[1:10,c(1,5:10)]
```

Now we plot the various measures from the resulting *node_measures data.frame* (output in Figure 156).

Plot *degree*, *degree_in*, and *degree_out* together (output in Figure 157).

Plot *degree* (*degree-in* + *degree-out*) and *betweenness* together (output in Figure 158).

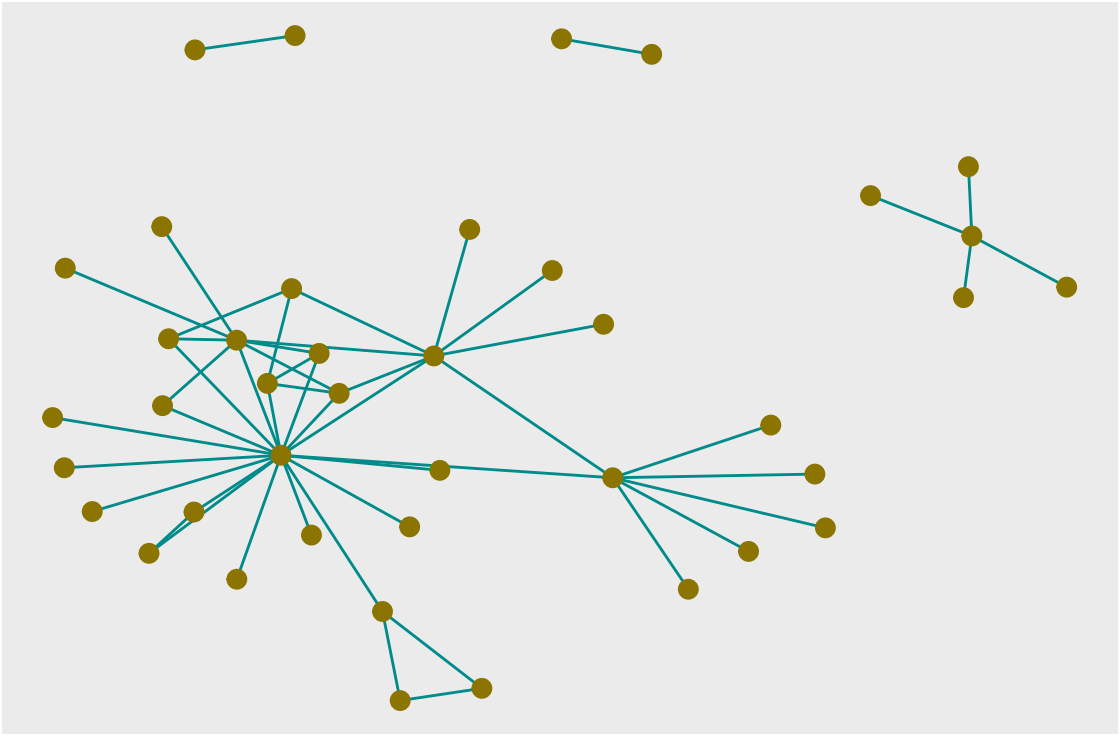


Figure 155: Adjusting node and edge plotting parameters

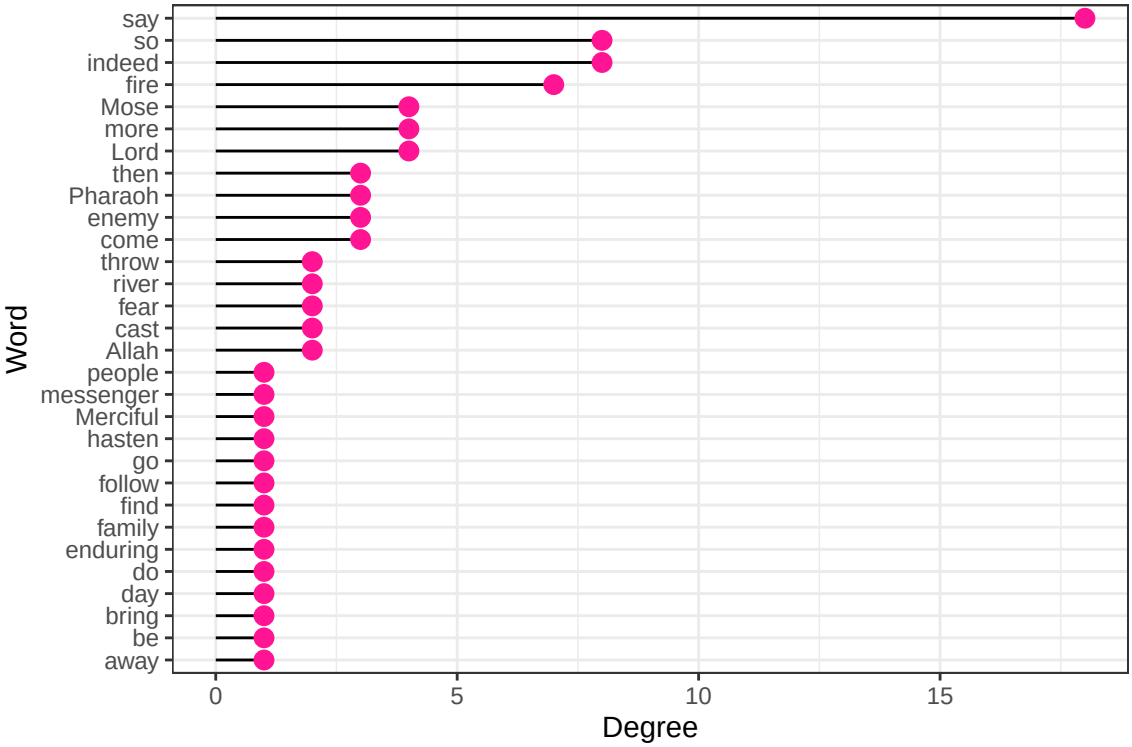


Figure 156: Degree for top 30 words

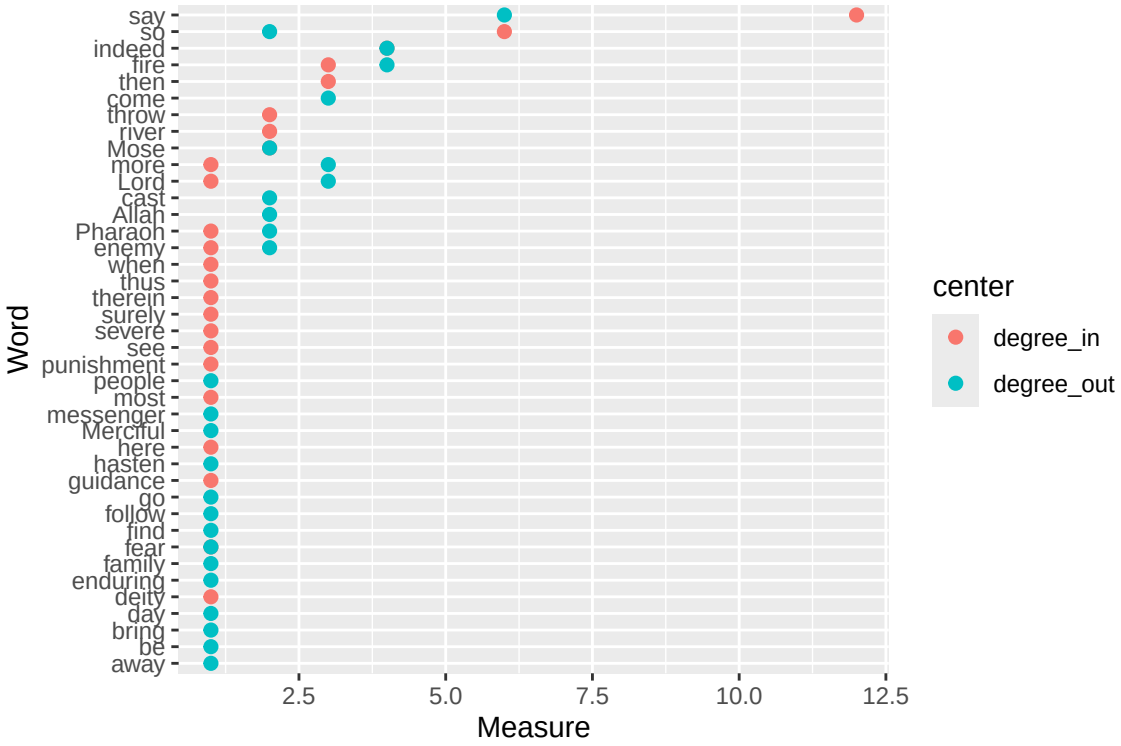


Figure 157: Degree-in and degree-out for top 50 words

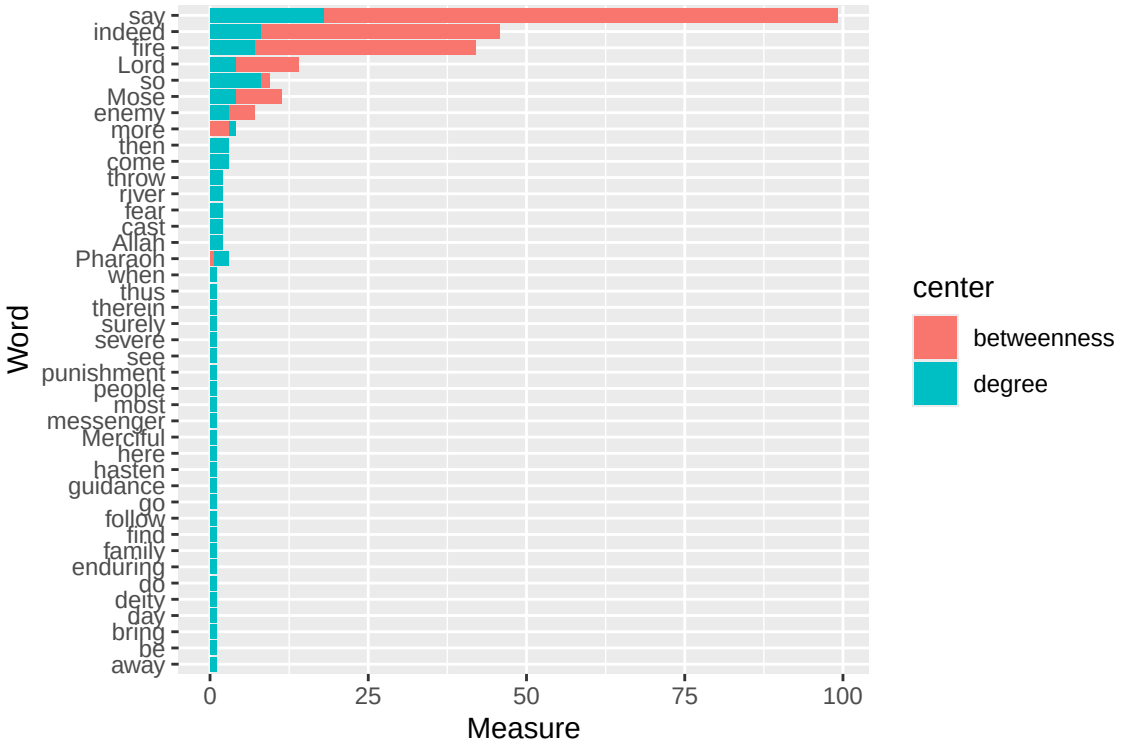


Figure 158: Degree and betweenness for top 50 words

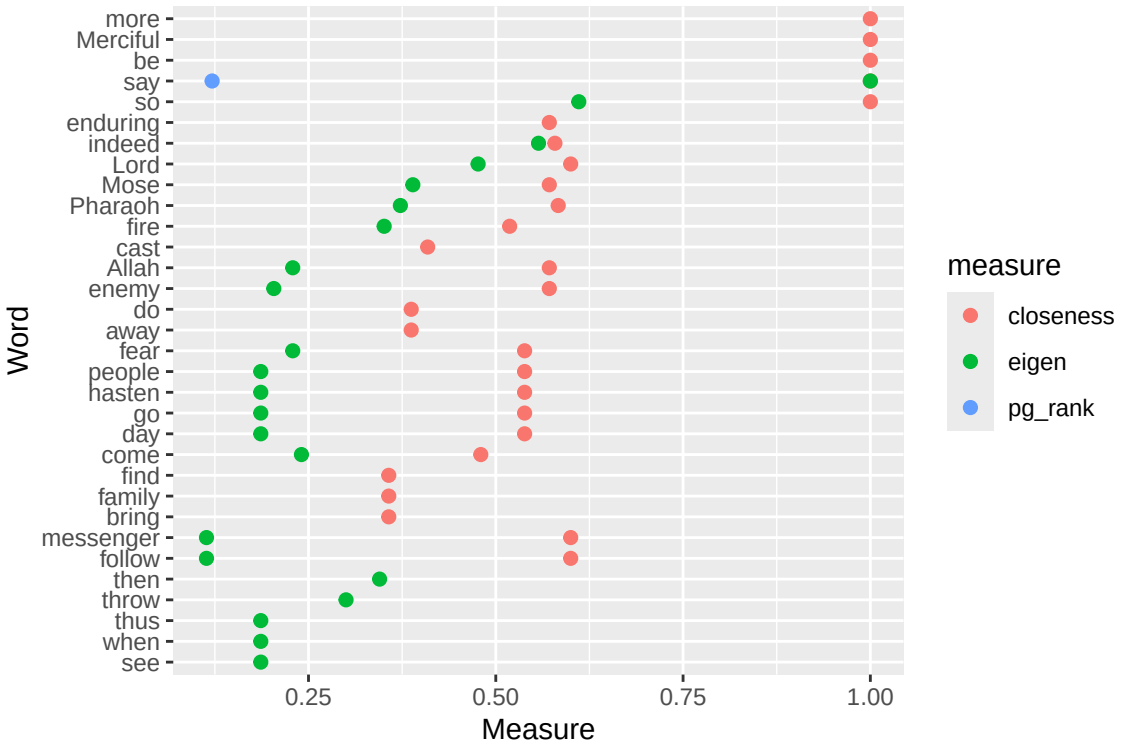


Figure 159: Other measures for top 50 words

Plot *closeness*, *pg_rank*, *eigen*, *br_score*, and *coreness* together (output in Figure 159).

Despite using *coord_cartesian(ylim = c(0, 1))* to scale the Measure coordinate, the values for br_score, and coreness are 0 or very small. Without any doubt "say", "Lord" together with "Allah" and "Mose" are the most influential and important words in Surah Taa-Haa.

Notice the slight difference in using the network measures with *tidygraph*. We can easily assemble the required measures for the nodes in a tidy *data.frame*. *tidygraph* has many functions that can give us information about nodes. We show examples of some measures that seem to measure slightly different things about the nodes.

- degree: Number of direct connections
- betweenness: How many shortest paths go through this node
- pagerank: How many links pointed to me come from a lot of pointed-to-nodes
- eigen centrality: Something like the page rank but slightly different
- closeness: How central is this node to the rest of the network
- bridge score: Average decrease in cohesiveness if each of its edges were removed from the graph
- coreness: K-core decomposition

6.5.3 Example combining selected node and edge measures from *tidygraph*

The following is an interesting example in true *tidyverse* fashion that combines some measures that we have not covered.[74]

We can also convert our active node or edge table back to a *tibble* and plot the output (see Figure 160).

For the next plot, we define our own specific colors. The center-most characters are in red and the distance to the center is the node size (see Figure 161).

[74] https://www.shirin-glander.de/2018/03/got_network/

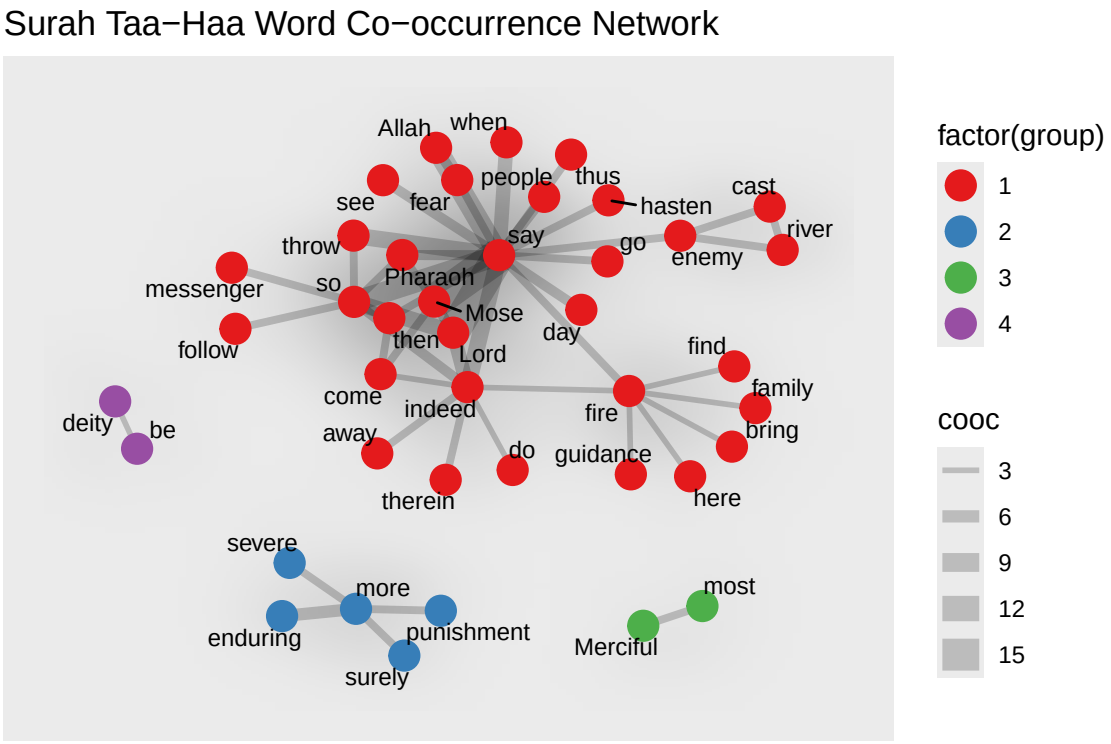


Figure 160: Nodes are colored by group

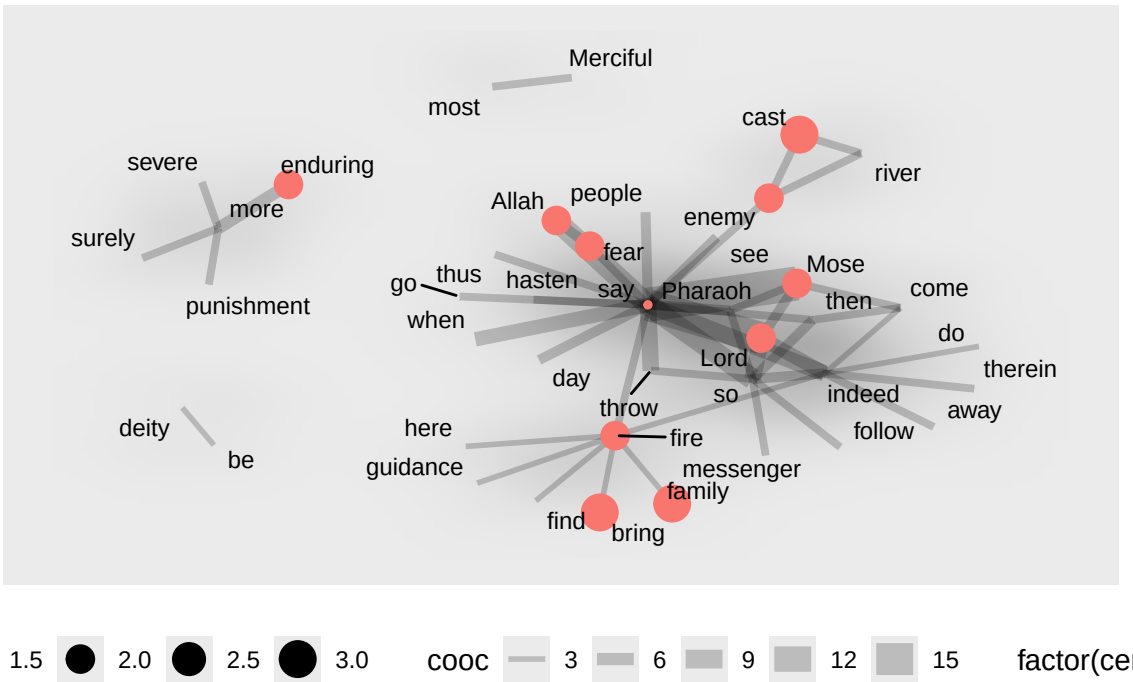


Figure 161: Nodes are colored by centeredness

Clearly, "say" is the *keyplayer* for the main group.

6.5.4 Who is the most important influencer?

In this section, we ask some questions about which is the "most important" node. We want to understand important concepts of network centrality and how to calculate those in R.

What is the most important word in this network? What does "most important" mean? It of course depends on the definition and this is where network centrality measures come into play. We will have a look at three of these (there are many more out there).

Degree centrality: Degree centrality tells us the most connected word: it is simply the number of nodes connected to each node. It can denote Popularity. This is often the only measure given to identify "influencers": how many followers do they have?. So far "say" has the highest, 18 (12 in and 6 out).

Closeness centrality: Closeness centrality tells us who can propagate information quickest. One application that comes to mind is identifying the superspreaders of infectious diseases, like COVID-19. "say" is no longer the highest.

Betweenness centrality: Betweenness centrality tells us who is most important in maintaining connections throughout the network: it is the number of times the node is on the shortest path between any other pair of nodes. It can denote brokerage and bridging. "say" is prominent here. As a Surah that narrates the story of Prophet Mose, that is understandable.

Eigenvector Centrality: Is a word (person) connected to other "well-connected" words (people)? It can denote connections. Again, "say" dominates.

Diffusion Centrality: Can a given word (person) reach many others within a short number of hops in the network? It can denote reach.

As we have seen, there is more than one definition of "most important". It will depend on the context (and the available information) which one to choose. Based on the previous plots, without any doubt "say", "Lord" together with "Allah" and "Mose" are the most influential and important words in Surah Taa-Haa. Indeed, it is about "Allah" narrating the true story of "Mose".

6.5.5 Build communities and calculate measures

We will do the analysis using *tidygraph*.

```
set.seed(123)
network_ego1 <- gt %>%
  mutate(community = as.factor(group_walktrap())) %>%
  mutate(degree_c = centrality_degree()) %>%
  mutate(betweenness_c = centrality_betweenness(directed = F, normalized = T)) %>%
  mutate(closeness_c = centrality_closeness(normalized = T)) %>%
  mutate(eigen = centrality_eigen(directed = F))
```

We can easily convert it to a *data.frame* using `as.data.frame()`. We need this to specify who is the "key player" in our ego network.

```
network_ego_df <- as.data.frame(network_ego1 %>% activate(nodes))
network_ego_df %>% slice(1:20)
```

Identify the prominent word in the network

We have converted the table_graph to a *data.frame*. The last thing we need to do is to find the top account in each centrality and pull the key player.

"Key player" is a term for the most influential nodes in the network based on different centrality measures. Each centrality has different uses and interpretations. A node that appears at the top of most centrality measures will be considered as the "key player" of the whole network.

```
# take 20 highest users by its centrality
kp_ego <- data.frame(
  network_ego_df %>% arrange(-degree_c) %>% select(name) %>% slice(1:20),
  network_ego_df %>% arrange(-betweenness_c) %>% select(name) %>% slice(1:20),
  network_ego_df %>% arrange(-closeness_c) %>% select(name) %>% slice(1:20),
  network_ego_df %>% arrange(-eigen) %>% select(name) %>% slice(1:20)
) %>% setNames(c("degree","betweenness","closeness","eigen"))
```

Top 10 words based on its centrality

From the table above, "say" tops in most centrality measures.

6.5.6 Visualize the network

We scale the nodes by degree centrality, and color it by community. We filter by only showing community 1 to 10 (see output in Figure 162).

```
network_ego1 %>%
  filter(community %in% 1:10) %>%
  top_n(100,degree_c) %>%
  mutate(node_size = ifelse(degree_c >= 1,degree_c,0)) %>%
  mutate(node_label = ifelse(closeness_c >= 0.001,name,"")) %>%
  ggraph(layout = "stress") +
  geom_edge_fan(alpha = 0.05) +
  geom_node_point(aes(color = as.factor(community), size = 1.5*node_size)) +
  geom_node_label(aes(label = node_label),repel = T,
                  show.legend = F, fontface = "bold", label.size = 0,
                  segment.color="royalblue", fill = "wheat") +
  coord_fixed() +
  theme(legend.position = "none")
```

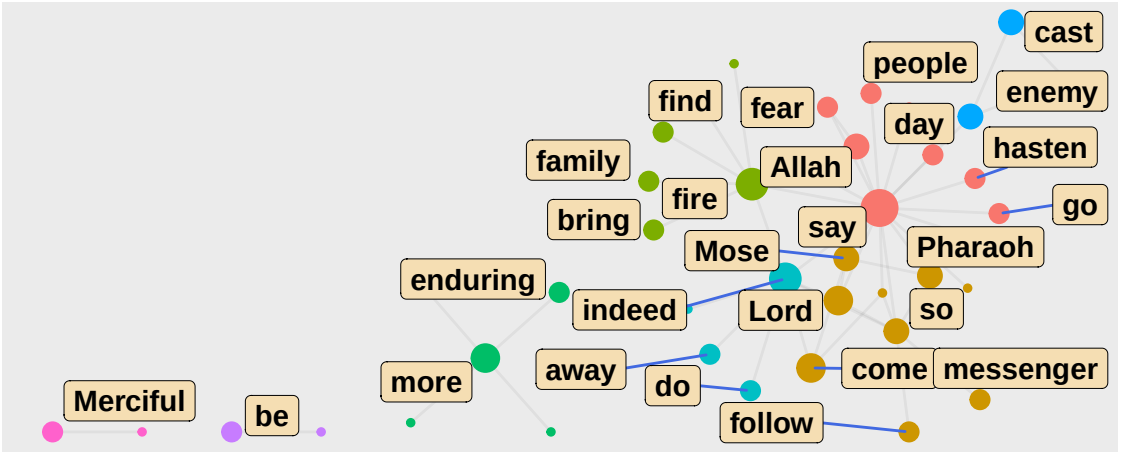


Figure 162: Top 10 word communities

ego() function

The neighbors of a specific node can be extracted with the ego() function. Below, we are looking for all words that are linked with "say", directly (order = 1) and indirectly (order > 1).

```
focusnode <- which(V(gd)$name == "say")
# ego(gd,order = 1, nodes = focusnode, mode = "all", mindist = 1)[1:7]
ego(gd,order = 2, nodes = focusnode, mode = "all", mindist = 1)[1:7]
# ego(gd,order = 3, nodes = focusnode, mode = "all", mindist = 1)[1:7]
```

We use this to test the small world and 6 degrees concept[75]. Here "say" reaches every other word after 2 degrees/orders.

6.5.7 Concentric layouts

We introduced some network graph layouts in Chapter 5. We discussed in more detail features about layouts, nodes, edges, and themes using the *ggraph* package where we used the same sample network of word co-occurrences in Surah Taa-Haa.[76] Here, we will show examples of new layouts.

Concentric circles help to emphasize the position of certain nodes in the network. The *graphlayouts* package has two functions for concentric layouts, *layout_with_focus()* and *layout_with_centrality()*.

The first one allows to focus the network on a specific node and arrange all other nodes in concentric circles (depending on the geodesic distance) around it. Below we focus on the word "Mose". However, it must be a connected graph. From previous plots, both *gd* and *gu* is not fully connected. So we focus on the largest cluster.

Taking the largest component

```
cld <- clusters(gd)
jg1 <- induced_subgraph(gd, which(cld$membership == which.max(cld$csize)))
jg2 <- simplify(as.undirected(jg1))
```

The parameter focus in the first line is used to choose the node id of the focal node (Mose = 1). The function coord_fixed() is used to always keep the aspect ratio at one (i.e. the circles are always displayed as a circle and not an ellipse).

The function draw_circle() can be used to add the circles explicitly (see the output in Figure 163).

```
got_palette = c("red", "blue", "green", "gold", "coral", "cyan4",
                "maroon", "deeppink")
focusnode <- which(V(jg1)$name == "Mose")
deg = degree(jg1)
ggraph(jg1,layout = "focus", focus = focusnode) +
    graphlayouts::draw_circle(col = "darkblue", use = "focus",
                              max.circle = 3) +
    geom_edge_link0(aes(edge_width = cooc), edge_color = "grey66") +
    geom_node_point(aes(size = deg), shape = 19, color = "hotpink") +
    geom_node_text(aes(filter = (name == "Mose"),
                       size = 2*deg, label = name),
                       family = "serif") +
    scale_edge_width_continuous(range = c(0.1, 2.0)) +
    scale_size_continuous(range = c(1,5)) +
    scale_fill_manual(values = got_palette) +
    coord_fixed() +
    theme(legend.position = "bottom")
```

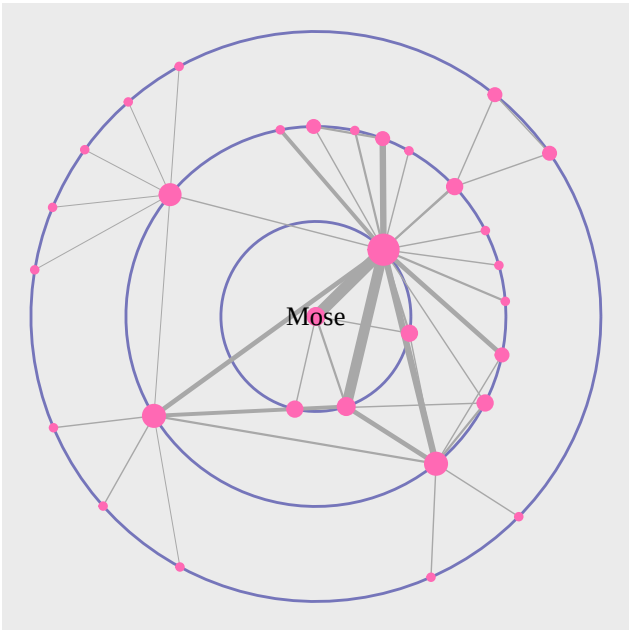


Figure 163: Using draw circle layout with Mose as focus node

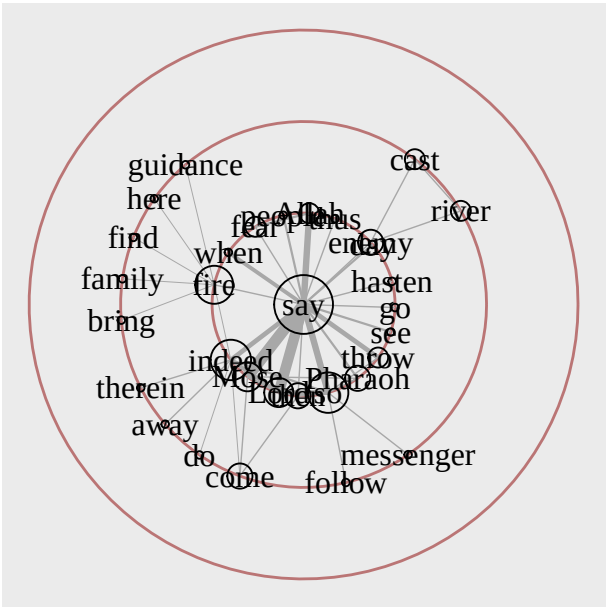


Figure 164: Same layout with say as focus node and text labels

Repeat with a change of the focus node and displaying all the words (see the output in Figure 164).

layout_with_centrality() works similarly. We can specify any centrality index (or numeric vector) and create a concentric layout where the most central nodes are put in the center and the most peripheral nodes in the biggest circle. The numeric attribute used for the layout is specified with the cent parameter. Here, we use the weighted degree of the characters. See the output in Figure 165.

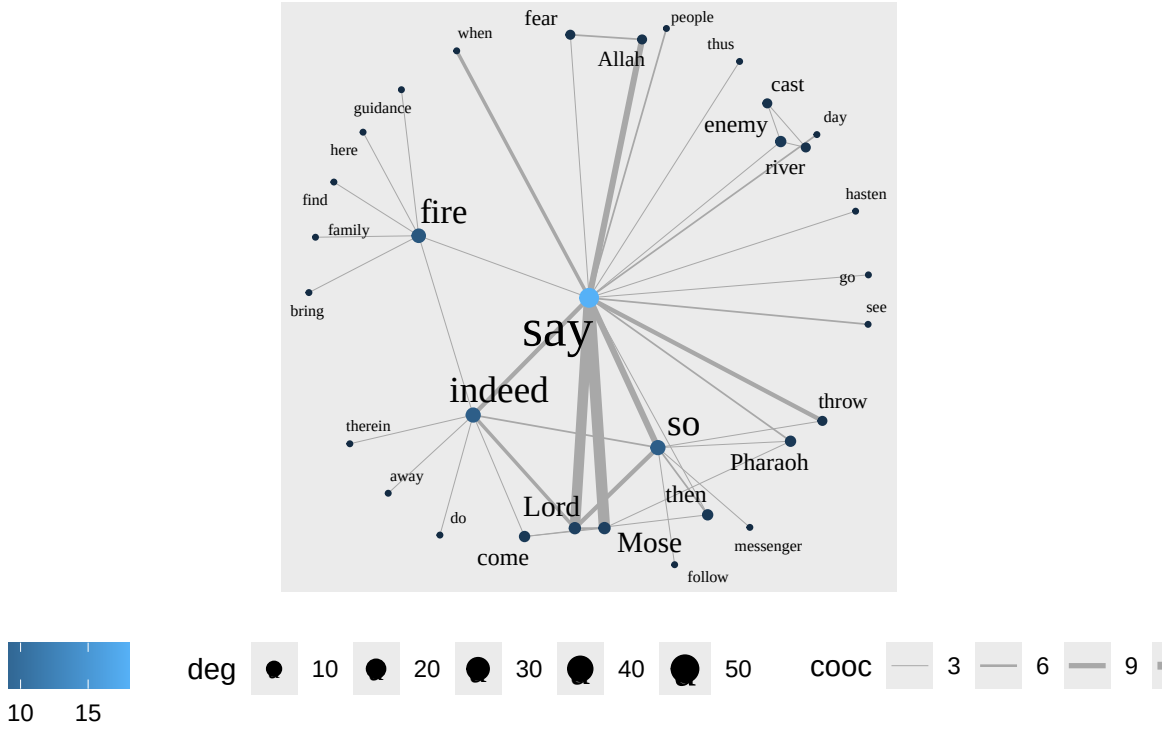


Figure 165: Weighted degree centrality layout

We repeat with *betweenness centrality* (see the output in Figure 166).

Stress layout and clustering

We focus again on *gd* and *gu*. Some clustering functions do not work on directed graphs. We show two different examples here. The first one is a directed network (outputs in Figure 167).

The second one is the undirected graph *gu* and *cluster_louvain* which does not work with directed graphs. *gu* does not have edge properties so we remove the aes(width=cooc). The output is in Figure 168.

Interestingly, the cluster functions give 4 for *gd* and 8 for *gu*.

Focus layout and clustering - focus on selected words

Earlier, we have shown how *layout_with_focus()* allows us to focus the network on a specific word and order all other nodes in concentric circles (depending on distance) around it. Here we combine it with clustering. The limitation is that it can only work with a fully connected network (see the output in Figure 169 and Figure 170).

layout_with_centrality() is based on a similar principle. We have shown this earlier. But here, we repeat with clustering (using *gu*) and look at the coreness centrality measure. Earlier, we have seen that *cluster_louvain()* gives different results than *clusters()* (see output in Figure 171 and Figure 172).

[75] http://networksciencebook.com/chapter/3#small-worlds
[76] https://rpubs.com/azmanH/708667

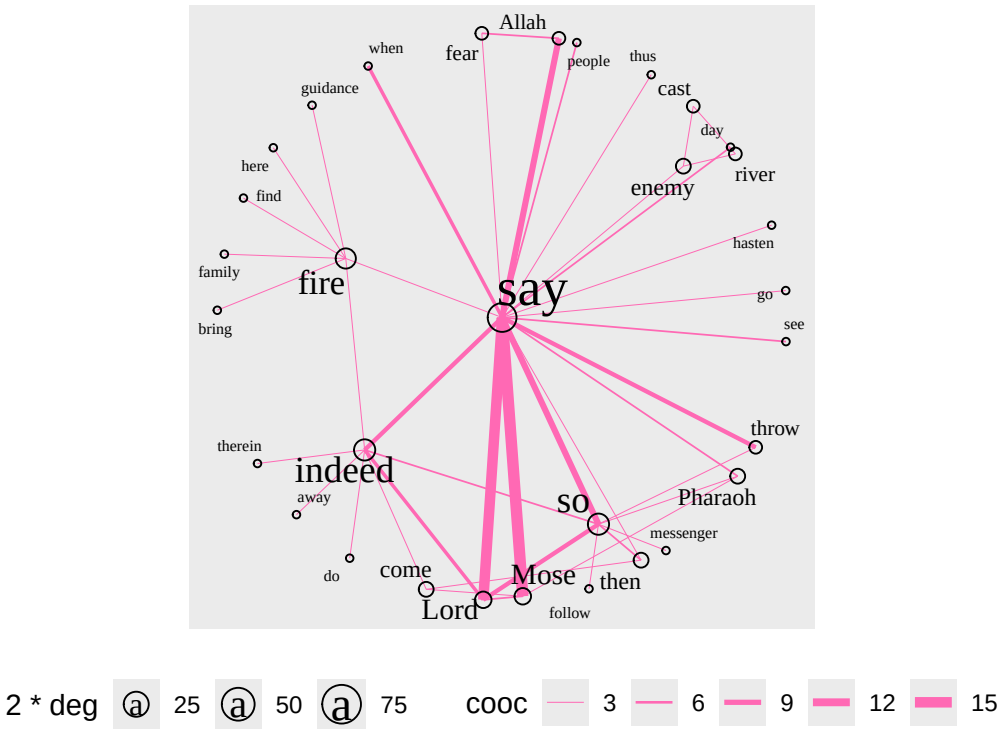


Figure 166: Betweenness centrality layout with degree for node size

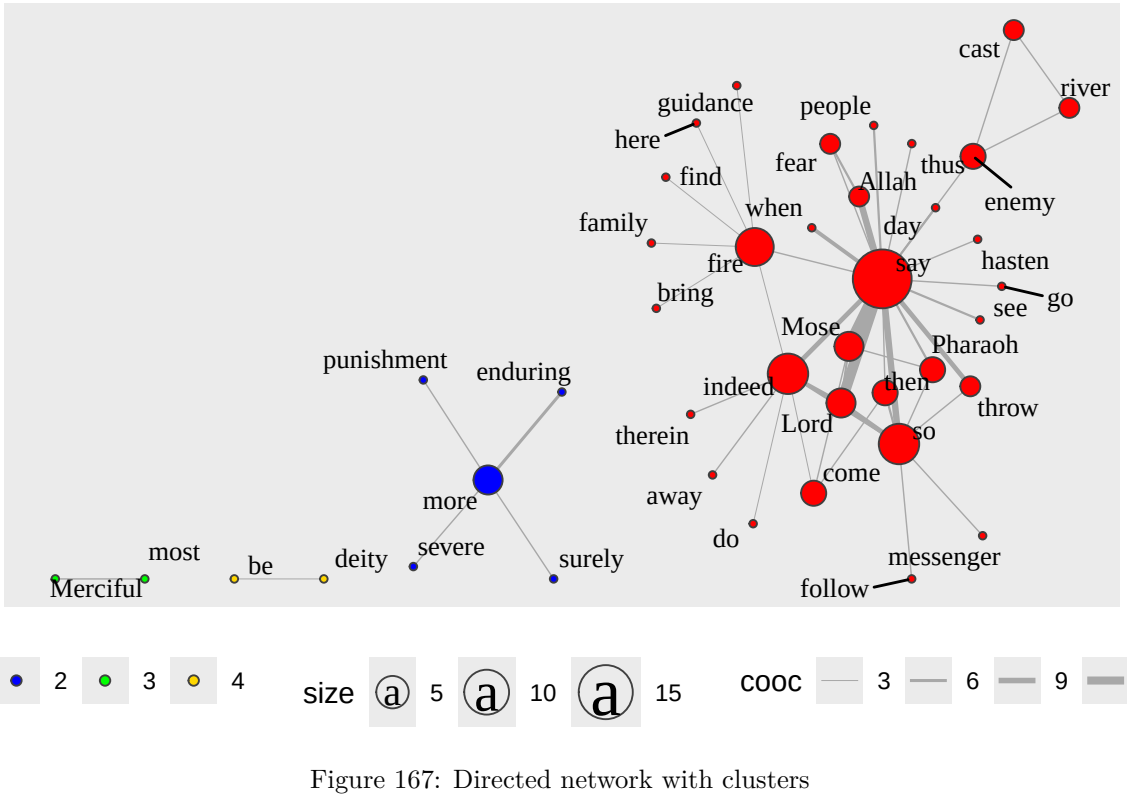


Figure 167: Directed network with clusters

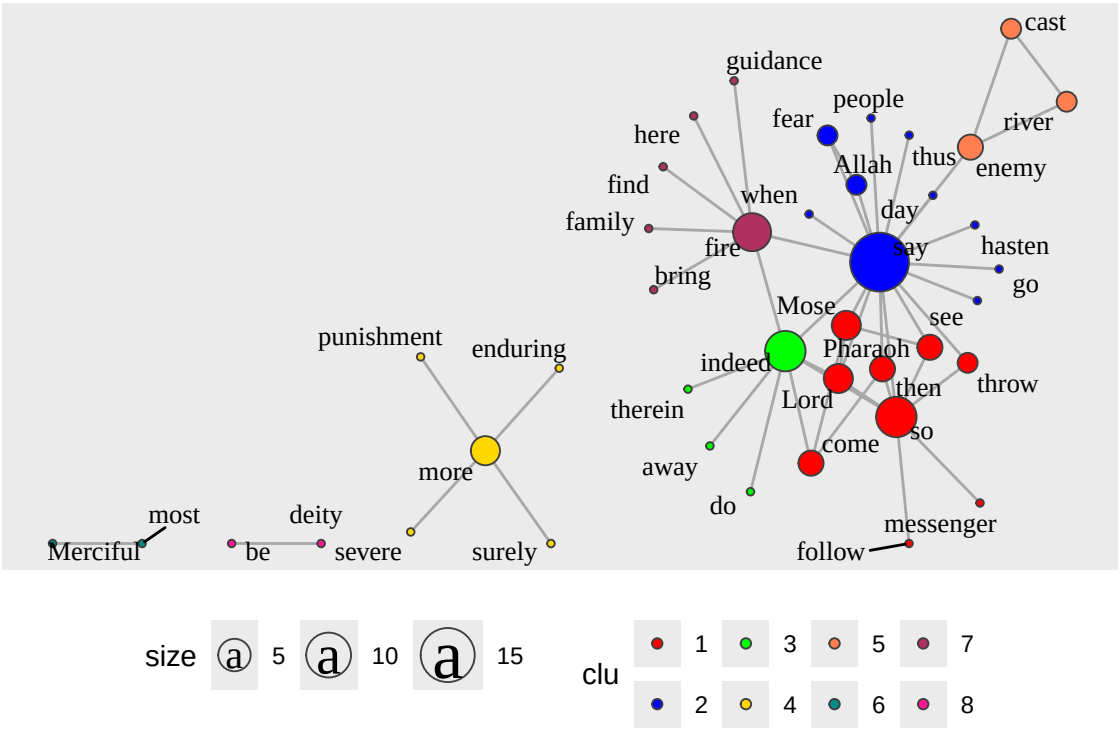


Figure 168: Undirected network with clusters

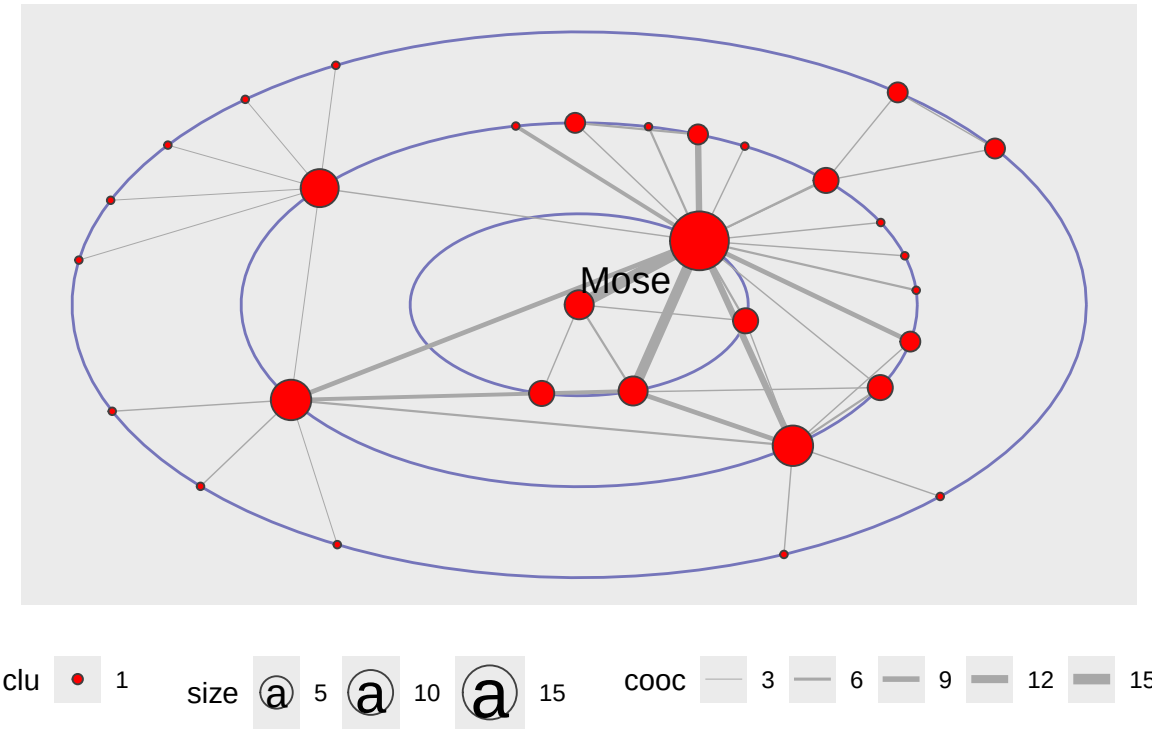


Figure 169: Focus layout with only the focus word

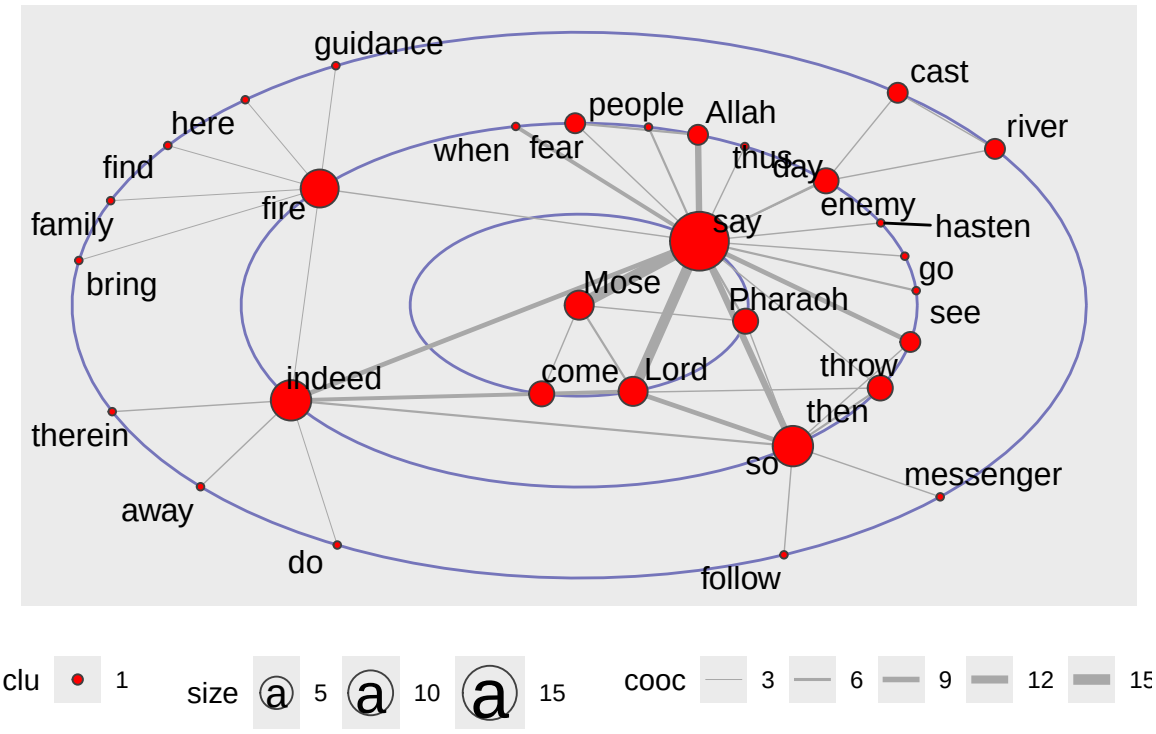


Figure 170: Focus layout with all words

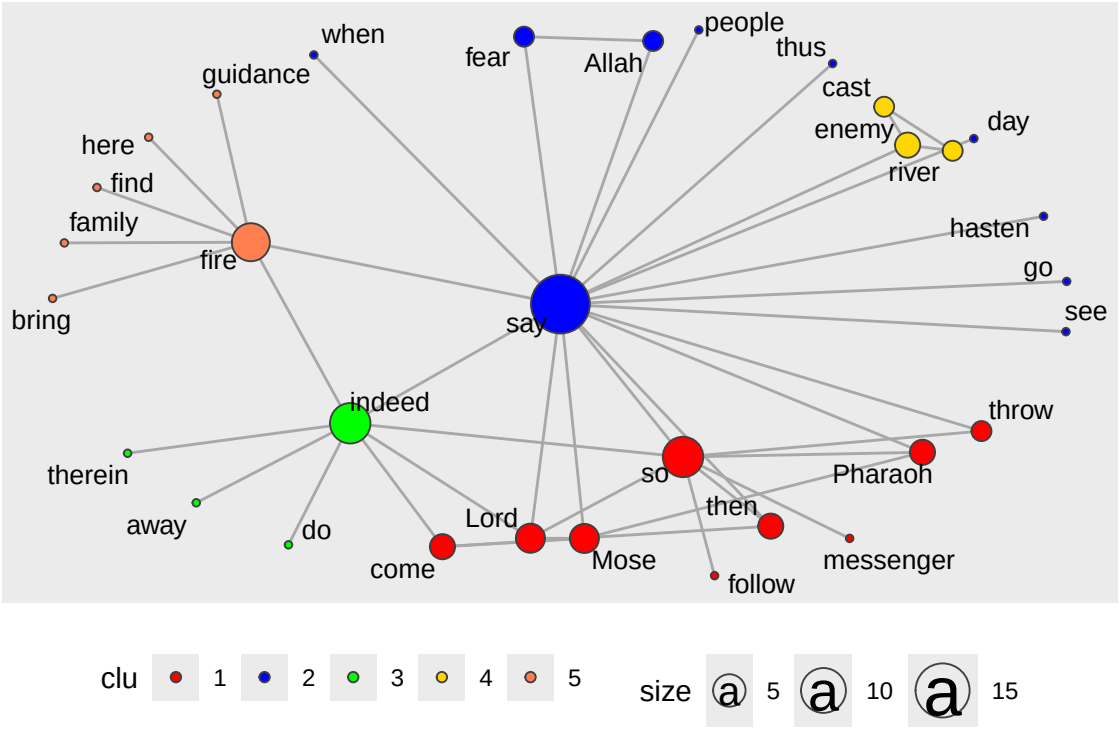


Figure 171: Centrality layout : graph.strength with clusters

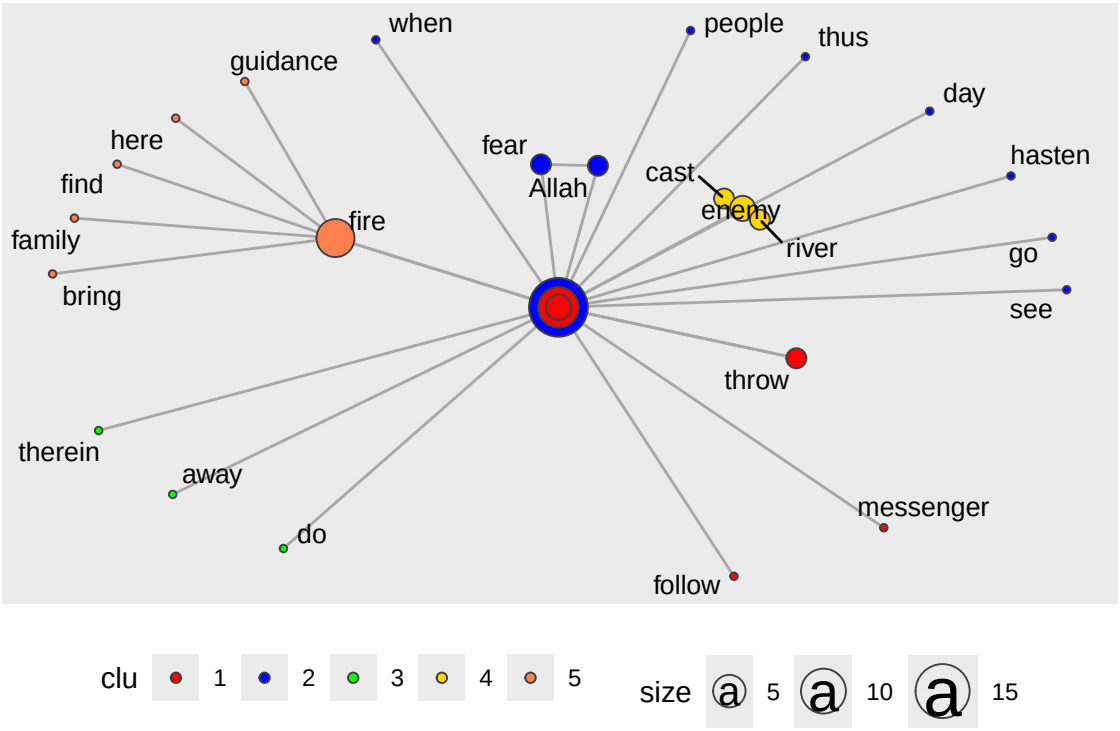


Figure 172: Centrality layout : graph.coreness with clusters

6.6 Summary

We have covered the characteristics of networks using our example of the word co-occurrence network from Surah Taa-Haa. We showed how to use the functions from *igraph* and *tidygraph* that measure these characteristics. We also showed different ways to use *ggraph* and its layout, node, and edge formats to visualize the network and its related measures.

The main objective is to understand the *position and/or importance* of a node in the network. The individual characteristics of nodes can be described by

- Degree
- Clustering
- Distance to other nodes

The *centrality* of a node reflects its influence, power, and importance. There are four different types of centrality measures.

- Degree - connectedness
- Eigenvectors - Influence, Prestige, “not what you know, but whom you know”
- Betweenness - importance as an intermediary, connector
- Closeness, Decay – ease of reaching other nodes

We summarize the use of some of these measures in simpler terms.

- Degree distributions help identify global patterns of networks.

- Clustering helps identify local patterns.

- Centrality measures determine the position and importance of nodes in networks.

We ended the tutorial with examples of using *ggraph* with the stress, layout_with_focus() and layout_with_centrality() functions from the *graphlayouts* package. These examples will be very useful in our future work.

In concluding, we refer to some interesting points about areas of research in networks.[77] Why Study Networks?

- Many economic, political, and social interactions are shaped by the local structure of relationships:
  - trade of goods and services
  - sharing of information, favors, risk
  - transmission of viruses, opinions
  - access to info about jobs
  - choices of behavior, education
  - political alliances, trade alliances
- Social networks influence behavior
  - crime, employment, human capital, voting, smoking
  - networks exhibit heterogeneity, but also have enough underlying structure to model
- Pure interest in social structure
  - understand social network structure

Primary Questions:

- What do we know about network structure?
- How do networks form? How do the efficient networks form?
- How do networks influence behavior?
- Diffusion, learning, peer effects, trade, inequality, polarization
- Dynamics, feedback

What are important areas for future research? Three areas for research include

- Theory
  - network formation, dynamics, design
  - how networks influence behavior
  - co-evolution?
- Empirical and experimental work
  - observe networks, patterns, influence

[77] https://www.coursera.org/learn/social-economic-networks

  - test theory and identify regularities

- Methodology
  - how to measure and analyze networks

From the items above,

- interest in and understanding the Quran words network structure, and
- empirical and experimental work to observe networks, patterns, and influence

are what we can relate to with our current work on Quran Analytics.

6.7 Further readings

Jackson, M. O. *Social and Economic Networks*, Princeton University Press, Princeton, New Jersey, USA, 2008 (Jackson, 2008).

Kolazcyk, E. and Csardi, G. *Statistical Analysis of Network Data with R*, Springer, New York,New York, USA, 2014 (Kolaczyk and Csardi, 2014).

Barebra, P., *Introduction to social network analysis with R* (http://pablobarbera.com/big-data-upf/html/02a-networks-intro-visualization.html)

Sadler, J, *Introduction to Network Analysis with R* (https://www.jessesadler.com/post/network-analysis-with-r/)

Text and Knowledge Modeling

Words are grouped, as a sentence, as a paragraph, like a chapter or part in a book, as a corpus, and across corpora. Words that exist in one group may appear in another grouping, or may not appear at all. For example, complex legal technical words will unlikely appear in a novel such as the Harry Potter series. Likewise, dramatic words are seldom used in a technical book like ours.

Groupings represent a higher dimension for words within texts. Hence any analysis must provide a holistic view of the word, within its context of groupings, whether it is in vocabulary sense, sense of grammar, semantical meaning, as well as pragmatical context.

Now we see words within language as an element of a complex system; which may at the same also complicated. The complex system here means a specific language is understood as a complex system and may be extremely complicated (depending on the language), and surely it is highly complex and highly complicated when viewed across languages and culture, and other dimensions such as psycho-linguistics.

In this part, we add two chapters that will be focused on Graph Theory (or Network Science) as the base methodology. In Chapter 7, we will focus on "text networks" where we study texts as a network of words. In Chapter 8, we then focus on "text classification models". The focus is on how the texts are "grouped" and what are the classifications for the groups. The final chapter of this part, Chapter 9, discusses the subject of Knowledge Graphs, where the focus is on the Tafseer of Imam Ibnu Katheer.

Some major unanswered questions are summarized at the end of the chapter as directions for future research.

7 Texts Network Analysis

Natural Language Processing (NLP) is a combination of linguistics and data science analyzing large amounts of *natural language* data which includes a collection of speeches, text corpora, and other forms of data generated from the usage of languages. The tasks of NLP vary from text mining and speech recognition (data-driven) to more complex tasks such as automatic text generation or speech production (AI-driven).

In this chapter we will focus on one particular aspect of NLP applied to a chosen text of the English translation of the Quran, namely *lexical semantic analysis*. This analysis focuses on what is termed as **individual words in context** analysis. Lexical semantics is the study of word meanings within its internal semantic structure or the semantic relations that occur within the corpus as a whole.[78] We will focus on the second approach, namely to study words in relation to the rest of the words in the complete text, in this case, the Saheeh English translation of Al-Quran.

We will also take a very specific approach by deploying network graphs (or properly known as graph theory). We start with visualization of words within the text as a network of relations (words in the text as *nodes* and their presence in a sentence as *directed edges*). The relations can take several forms to suit a particular interest. It may be to discover main messages in the text or analytical reasoning such as to uncover the major topics within those messages. It can be explorative analysis, such as how these messages and topics relate to each other within the main message (or text).

Another important point to mention is the difference between the *parametrical* and *non-parametrical* approach to the task at hand. The parametrical approach relies on some pre-built models, such as *sentiment scoring*, *semantic ontologies*, etc. The non-parametric approach does not rely on any models and instead will be driven by the empirical nature of the words and the text itself (i.e. do not rely on other samples from outside of the sample at hand). We will use the second approach by using network analysis and graph theory.

In network analysis, identifying a few methods will help greatly. An example is the formation of the network whether it follows a random graph or any particular *graph structure*. Another important issue is on the *emergent structures*, whether any emergent structure can be observed, and if it exists, we can uncover the factors of the emergent structures and sub-structures. This will bring us into the subject of *complicatedness* and *complexity* of *systems analysis*.

Given the enormous possibilities and size of the task, this chapter will focus on providing preliminary findings using basic network analysis. We will identify some open issues for future work.

To perform the various analysis, we will use two main packages in R, namely *quanteda* (Benoit et al., 2018) and *igraph* (Csárdi, 2020). Quanteda is a complete suite of R packages for text analytics with many ready-made built-in functions that are easy to use.[79] iGraph is a network (or graph network) package in R.[80] Both packages are well developed and supported within the R programming community. For the data, we will use the prebuilt text in *tidydata* format from *quRan* package, and for some of the utilities required, we will use the *quanteda* package.

For purposes of fast computation and visualization of a large network, we will use open-source software, *Gephi*[81]. Similar software are *Pajek*[82], *Cytoscape*[83], and *NodeXL*[84]. As far as computation is concerned there are no additional advantages offered by these software applications, except for easier visual manipulations and production of images. We will rely on Gephi for this purpose while using **R** as our main engine for computations.

[78] Lexical semantics, Oxford research encyclopedias; https://oxfordre.com/linguistics/view/10.1093/acrefore/9780199384655.001.0001/acrefore-9780199384655-e-29

[79] Quanteda reference, https://quanteda.io/index.html

[80] iGraph reference, https://igraph.org/r/

[81] https://gephi.org

[82] http://pajek.imfm.si

[83] https://cytoscape.org

[84] https://www.smrfoundation.org

7.1 A brief on *quanteda*

The work for this chapter and the next will rely on the *quanteda* package; hence we feel it is appropriate to present a short and brief tutorial on the package (as we have done for *tidytext*, *igraph*, and *ggraph* in earlier chapters).

The *quanteda* package is among the recent introductions into the family of NLP tools in **R**. It was developed by Kenneth Benoit and many developers, supported by the European Research Council(Benoit et al., 2018). The package contains comprehensive text modeling functions from start to end - which includes basic features from *tokenization* to *unsupervised learning models* of text analysis.[85]

quanteda has four basic types of objects:

1. *corpus()* which is corpus, the collection of texts in textual format.
2. *tokens()* which is a tokenizing function, the tokens for the texts with all the metadata and tagging
3. *dfm()* a Document-feature matrix (DFM), a document-term-matrix, which is a sparse matrix of 0 and 1, indicating the occurrence of a term in a document (which is a sentence or set of texts)
4. *fcm()* a Feature Co-occurrence Matrix (FCM), a term-to-term-matrix, which is a sparse matrix of 0 and 1, indicating co-occurrence of a term with another term, within the entire document or corpus.

In most text analytics packages, the first three objects are available directly from the package, while the feature's matrix is generated from pre-processing exercises before being fed onto a learning model. The advantage of *quanteda* is fast and seamless processing or creation of the FCM.

Text analysis with *quanteda* goes through all these four types of objects either explicitly or implicitly.

corpus()

Once a *corpus* is created, we can manipulate the corpus with many functions. *corpus_reshape()* is an important tool for converting data from long to wide format. Subsetting of a corpus is done via *corpus_subset()*. *corpus_segment()* allows us to select documents in a corpus based on document-level variables. Trimming the corpus is through *corpus_trim()*.

tokens()

To create tokens from a corpus or text data, we use the *tokens()* function, which will create a tokenized dataset, consisting of tokens object. Once we have the tokens object, we can manipulate it with many ready-made functions, such as *tokens_lookup()* for quick search of tokens, *tokens_subset()*, *tokens_sample()*, *tokens_split()* for creating subsets the tokens; _tokens_select(), *tokens_replace()*, which are an important tool for replacing or changing items inside the tokens object; and finally, *tokens_remove()*, *tokens_tolower()*, *tokens_wordstem()* for manipulating the tokens object.

char()

There are also character-level manipulation functions that serve general utility purposes. These are: *char_tolower()*, *char_toupper()*, *char_ngrams()*, *char_select()*, *char_remove()*, *char_keep()*, *char_trim()*, etc. They are all are similar to the methods used in *tidytext* character manipulations functions.

dfm()

Document Feature Matrix object is created by *dfm()* function. This is among the major objects in *quanteda*. It is a sparse matrix with named rows and columns. The names for the rows are the document labels, and the names for the columns are the features (or tokens). Once created, we can use *dplyr* like functions such as *dfm_group()* for group-by, *dfm_select()* for selecting tokens, *dfm_remove()* for removing tokens, *dfm_sort()* for sorting, and so on.

fcm()

[85] For general reference, readers should refer to the quanteda documentation and tutorials at https://quanteda.io/index.html

Feature Co-occurrence Matrix object is created by *fcm()* function. It is a square sparse matrix with named rows and columns. The names for the rows and the columns are the features (or tokens). Once created, the manipulations are similar to the *dfm* object, where we can use functions such as *fcm_select()* for selecting tokens, *fcm_remove()* for removing tokens, *fcmm_sort()* for sorting, and so on.

Utility functions

There are a few useful functions to apply on *dfm* or *fcm* objects, which are handy, such as *docfreq()* for calculating frequencies of the tokens, and *topfeatures()* for getting the top features. General ones include: *ndoc()*, *nfeat()*, *nsentence()*, *ntoken()*, *ntype()* for counting purposes. One of the less well-known, but powerful functions is *dictionary()*, which creates a dictionary type data, which is handy for dealing with large amounts of text data, when generating vocabulary is required. The *dictionary* object can be manipulated easily with *dictionary_edit()*, *char_edit()*, and coercion between objects is by using the *as.dictionary()* function.

Network and plotting functions

Creating and converting objects into networks, and plotting is a cumbersome process. *quanteda* makes this easy by a seamless process of converting *dfm* or *fcm* object into an *igraph* object, which then can be used for graph manipulations and calculations. It also creates few wrappers around the *ggplot2* functions for easy plotting of network objects. An example of such functions is *textplot_network()*.

statistical functions

The statistical functions are named as *textstat_xxx*(). They are *textstat_simil()* for texts similarity calculations, *textstat_dist()* for texts distance measures, *textstat_frequency()*, *textstat_keyness()*, *textstat_collocation()*, *textstat_entropy()*, and numerous others. All of these functions are extremely useful for quick calculations of the statistical measures which are available for usage in analysis.

quanteda textmodels

As an extension to the brief on *quanteda*, we introduce the *quanteda.textmodel* package. It is a dedicated suite for text modeling work, which has many applications for running *statistical learning models* - taking advantage of the data structure of *quanteda* objects. There are many models developed include *textmodel_wordscores()*, *text_model_affinity()*, *textmodel_svm()* (Support Vector Machines), *textmodel_nb()* (Naive Bayes), *textmode_lsa()*, and others. We will describe the models in the next chapter (Chapter 8).

R code examples of using *quanteda* are given below:

```
library(quanteda)

# create a corpus
corp_kahf <- corpus(quran_en_sahih)
corp_kahf_sub <- corpus_subset(corp_kahf,ayah >= 100)

# tokenize and process
toks_kahf <- quanteda::tokens(corp_kahf,remove_punct = TRUE) %>%
              tokens_tolower() %>% tokens_remove(pattern = stop_words$word,
                                                 padding = FALSE)

# create dfm and fcm using tokens or corpus
dfm_kahf <- dfm(toks_kahf); dfm_kahf <- dfm(corp_kahf)
fcm_kahf <- fcm(toks_kahf); fcm_kahf <- fcm(corp_kahf)

# dfm and fcm manipulations
dfm_kahf_sub <- dfm_subset(dfm_kahf, surah_title_en == "Al-Baqara")
fcm_kahf_sub <- fcm_select(fcm_kahf, pattern = c("allah","lord"))

# using dictionary
dict <- dictionary(list(god = c("allah","lord"),
                prophet = c("prophet","muhammad","moses")))
dfm(corp_kahf, dictionary = dict)
```

For the rest of this chapter, we will utilize the *quanteda* package and work through our analysis utilizing the various functions for the dual purpose of providing a tutorial and applying them to specific examples.

7.2 Analyzing word cooccurrence as a network

7.2.1 Network dynamics: growth of word co-occurrence network

In network analysis, an important aspect is the dynamics of the network growth, from a few nodes and edges until it becomes a full-blown network. This phenomenon is important in network analysis, whereby a network may start as a small set, and over time it grows into its full size. In the words network, it may start with a few central words (or vocabulary) and as more words are added to the network, it will grow into a full network of words.

For the Saheeh English Quran, how does this growth behavior look like? This is the question we want to investigate. From the start, we know that the most frequent (and hence central word) is "Allah". How do other words start to attach to this central "node", as we increase the words by the order of frequencies?

```
quran_all = read_csv("data/quran_trans.csv")
tokensQ = quran_all$saheeh %>%
      tokens(remove_punct = TRUE) %>%
      tokens_tolower() %>%
      tokens_remove(pattern = stop_words$word, padding = FALSE)
dfmQ = dfm(tokensQ)
fcmQ <- fcm(tokensQ, context = "window", tri = FALSE)
fcm_tpnplot = function(fcmQf,n,vls){
  feat <- names(topfeatures(fcmQf, n))
  fcmQ_feat <- fcm_select(fcmQf, pattern = feat)
  v_size = rowSums(fcmQ_feat)/min(rowSums(fcmQ_feat))
  fcmQ_feat <- fcm_select(fcmQ, pattern = feat)
  fcmQ_feat %>%
    textplot_network(min_freq = 0.5,
                edge_color = "gold",
                edge_alpha = 0.5,
                edge_size = 2,
                vertex_size = 1,
                vertex_labelsize = vls*log(v_size))
}
p1 = fcm_tpnplot(fcmQ, n = 10,vls = 2)
p2 = fcm_tpnplot(fcmQ, n = 20,vls = 1)
p3 = fcm_tpnplot(fcmQ, n = 50,vls = 0)
p4 = fcm_tpnplot(fcmQ, n = 200,vls = 0)
cowplot::plot_grid(p1,p2)
```

```
cowplot::plot_grid(p3,p4)
```

Figure 173 shows the network growing from 10 to 20 words; and Figure 174 is from 50 to 200 words.

We can observe that the word network grows centrally from the single word "Allah", and it grows in a particular way as more words are added until the network is dense. More importantly, the whole word network (for co-occurrence network) is "one" single large network, densely organized. This is an important observation. The question is what does it mean?

Here we enclose a sample of a full-blown word co-occurrence network from the novel Moby Dick (Chapter 1) as a comparison (in Figure 175). It is a fully connected network, forming a single large network, but the

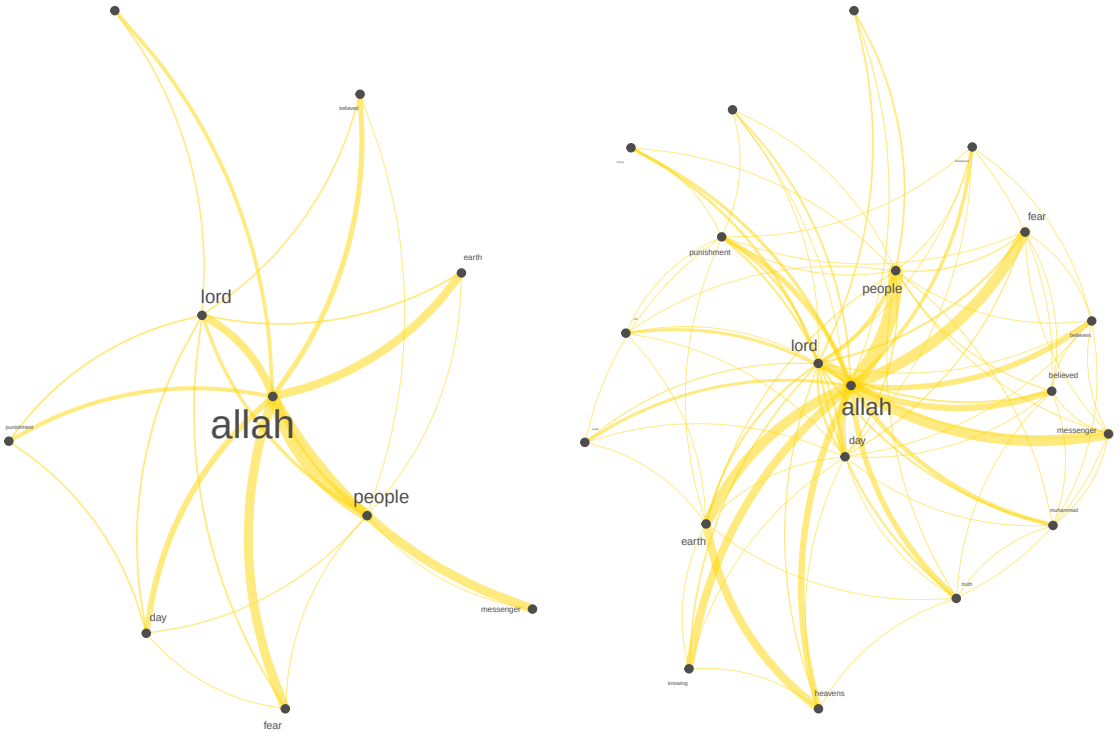


Figure 173: Growth of words co-occurrences network in Saheeh

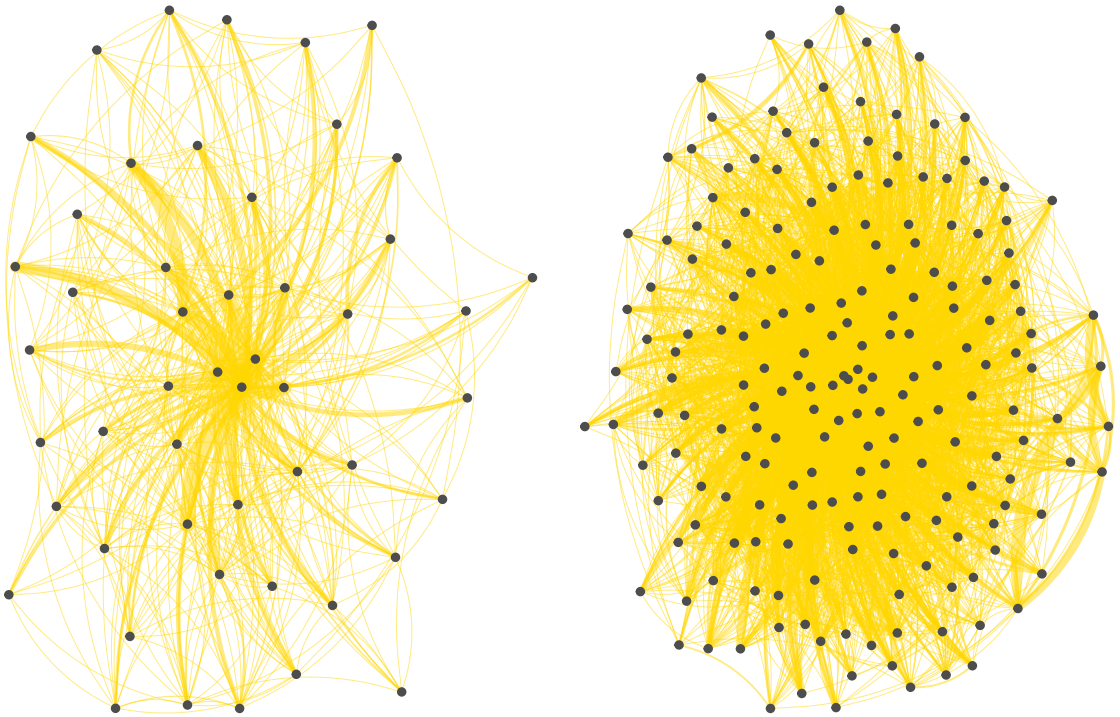


Figure 174: Growth of words co-occurrences network in Saheeh

network is not as dense as the network shown for Saheeh. In fact, if we do a similar step of checking the growth of the network, it is not the same as what we see in Saheeh.[86]

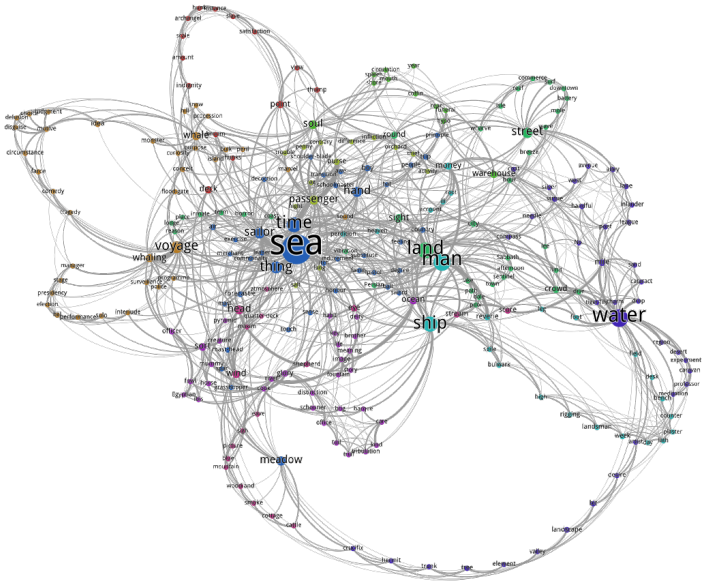


Figure 175: Example of co-occurrence network in Chapter 1 of Moby Dick novel

Furthermore, a detailed look at the plot (which is not shown here) reveals that the early keywords seem to have some "themes" to it; namely about "allah", "lord", "believe", "day", "people", "muhammad", "messenger", and the themes grow out of these main themes. While these themes grow, the centrality of "allah" remains and grows stronger as the network size expand. This is termed the "emergent structure" of the network. Why this is true, is a subject that requires further research and analysis, which we encourage readers to pursue.

And if we expand to all co-occurrences on the entire Saheeh corpus, we will get the picture in Figure 176[87], which is amazingly interesting.

The center of the network remains singly to "allah", and a zoomed view to the center is shown in Figure 177, which shows the central node, and all other major nodes (i.e., themes) - which are ordered as "allah", "lord", "people", "day", and so on. These "themes" interestingly coincide with the major "subject matters" as discussed by one of us through a qualitative analysis of the Quran.

7.2.2 Word co-occurrence network statistics

The word co-occurrence network is about understanding how each word that appears in the text relates to all other words which appear in the whole text. The connections or links between the words explain the structure of the messages or topics of the texts. The example in Figure 175 is a word co-occurrence network for Chapter 1 of Moby Dick's novel. It shows the whole text is centered around "sea" and "man", and sub-grouped by "water", "ship", and "voyage". The colors of the nodes represent sub-groupings (or cliques) whereby such sub-groupings may represent another message or sub-topic by themselves. The same process happens in growing the network shown in Figure 175 for Saheeh's entire corpus.

Now let us work using the *igraph* package and explore various statistical analyses using graph theory in understanding the network.

[86] We do not enclose the plots here to save space. Readers can repeat the same exercise using the enclosed code to check the results for themselves.

[87] The figure is obtained from Gephi; since plotting a large network of this size is not efficient in **R**

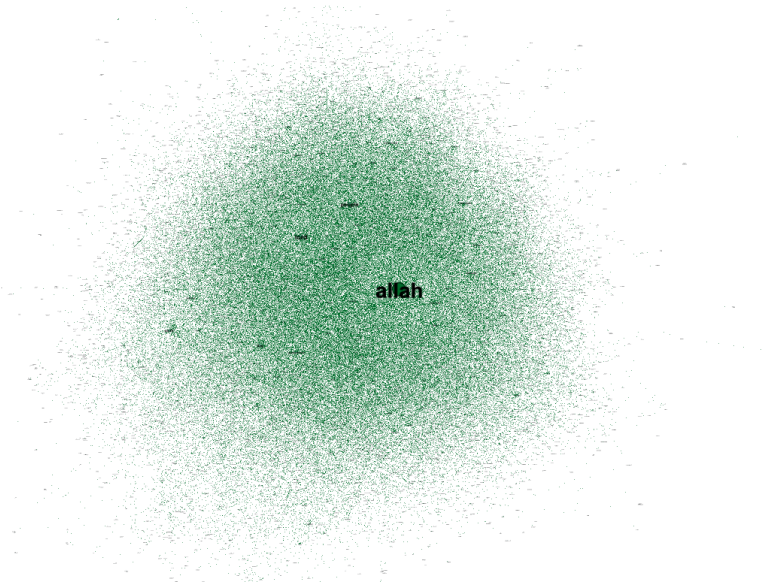


Figure 176: Saheeh entire corpus word co-occurence network

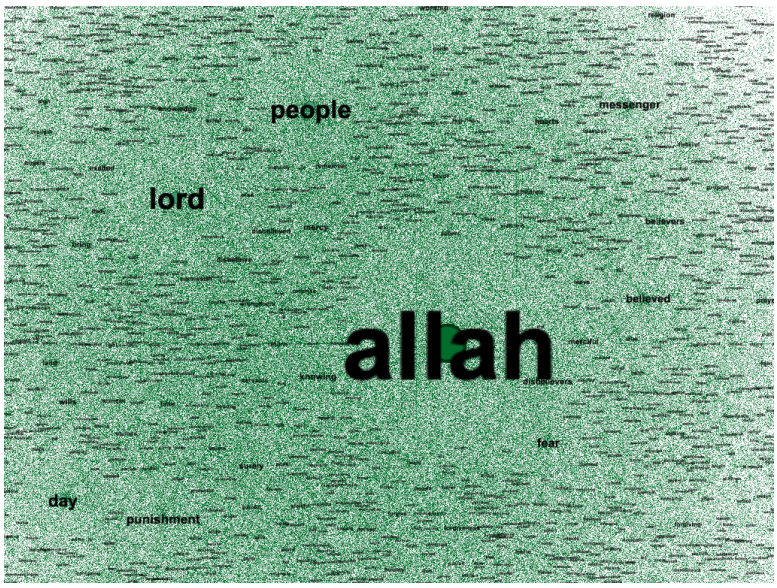


Figure 177: Close up view of the center of the network

```
library(igraph)
igphQ = quanteda.textplots::as.igraph(fcmQ)
```

First let us get the number of words (nodes) and co-occurrences (edges) in the whole word co-occurrence network (graph).

```
paste("nodes =",length(V(igphQ))," and edges =",length(E(igphQ)))
```

There are 4,801 words (nodes) and 267,668 edges in the network. Note that we have removed all the stopwords, which otherwise will confound the network with all the stopwords in between. Recall that from Chapter 5, we have recorded that the total unique tokens (words) for Saheeh is 5,739 (including stopwords). Almost one thousand tokens are not present, due to either being removed as stopwords or the tokens have zero co-occurrence, and hence not included.

There are a few general statistics that we are interested in, namely: *diameter*, *paths* and *distance*, *connectedness*, *clustering*, and *modularity* measures of the network. The previous chapter introduced a basic tutorial on these measures using a smaller word co-occurrences network graph from just one Surah. We now extend applying the same concepts for the entire Saheeh Quran corpus.

7.2.3 Diameter and average distance

In a network, the diameter is the measure of the "longest span", which implies the maximum "hops" or "steps" it takes from one node to reach the furthest node from it. This is also called the measure of "small-world properties" of Watts-Strogatz.[88] In a word network, it means how many words in between that it will take for a word to be connected to another word.

So how many words in between, for it to be connected to the furthest word in Saheeh? The answer is 6 words. How do we make sense out of this number? As a comparison, the network diameter for the internet is 6.98 and the E.coli metabolism network has a diameter of 8. Comparing with English texts (such as novels or textbooks) the "normal diameter" is between 10 to 14, and the measure is about the same for a few other Latin-based languages.[89] What we are observing here is a phenomenal structure, a corpus of English text having a measure of diameter much smaller than any normal texts, and extremely close to the measures of the network in nature (e.g., E.Coli protein network).

Average distance is a measure of "halfway" steps needed to reach the "center" of the network. The measure for Saheeh is 2.55. The comparable number for English books is between 3.33 to 3.60. Again, in our case here we can say that any word is not more than 2.55 steps away on average from the center word, which is "Allah", the central theme.

What all of these measures mean is that the word network for Saheeh exhibits similarities to the Small World network! In fact, for an Erdos-Renyi (random graph) network, the comparable numbers are between 14 to 20 for the diameter and between 4.60 to 6 for the average distance.(Ban et al., 2014) This means that the words co-occurrences in Saheeh are not random in nature, and in fact well structured as a dense network.

These properties very strongly indicate how closely related are all the words in Saheeh and the conciseness of the sentences in the text. Lower diameters and average distances are good measures of "efficiencies" in a large network. In the case of Saheeh, the measures clearly imply that the words in the texts are used with extreme efficiency.

7.2.4 Connectedness

Measures of connectedness reflect the components of the network, whether the network consists of a single large component or separated into few components. The codes below compute these measures:

[88] Please refer to https://en.wikipedia.org/wiki/Watts–Strogatz_model, for a quick guide.
[89] Ban et al. (2014)

```
comp_size = components(igphQ)
comp_size$no
comp_size$csize
```

The result shows that there are 9 components, and in fact the single largest component (giant component) consists of 4,783 nodes, which is 99.63% of the total nodes. This shows that the network is actually a single giant component, and it is also a *fully connected* network, which means that there is no single word that is not related to at least another word. It also implies that the whole network (i.e. every word) has a relation (directly or indirectly) with every other word in the network.[90]

7.2.5 Degree distributions

A degree is a link between two words, and the total number of links attached to a word is called the degree of the node (word). We are interested how does the degree for the entire nodes in the network loos like, from a statistical distributions perspective.

The codes below compute the degree for the Saheeh network and plot them.

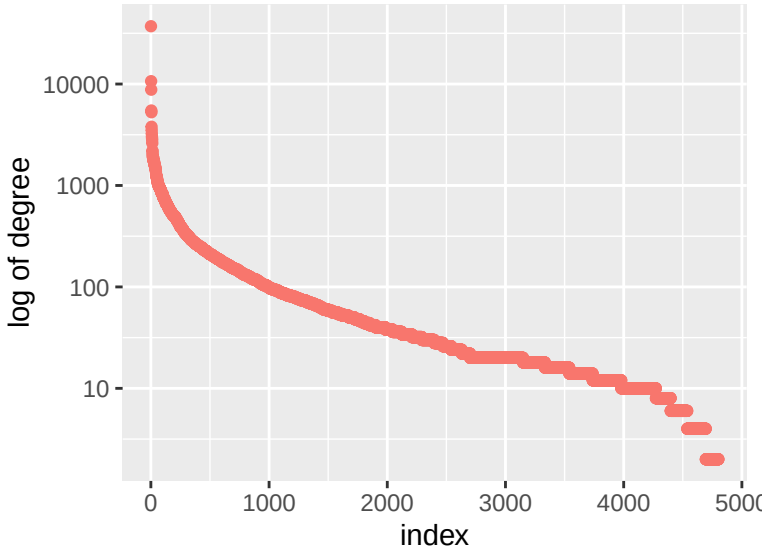


Figure 178: Plot of the degree distribution for Saheeh

Figure 178 looks similar to the tf-idf plot for Saheeh in Figure 2.14 from Chapter 2 - which indicates that the degree follows Zipf's law, and hence is distributed following Power Law distributions. It shows the case where a very small number of words have a high degree of edges, whilst a very large number of words have an extremely small number of edges.

We show a simple demonstration what the degree means by printing the top twenty words ranked by its degree in the following texts, which is autogenerated by the codes written for this book:

allah, lord, people, day, earth, punishment, fear

messenger, believed, muhammad, truth, heavens, knowing, believers

life, disbelievers, fire, mercy, surely, moses

It is forming like a sentence, saying "Allah (the) Lord (of the) people, day (on) earth, punishment fear, messenger believed, Muhammad truth". Actually, when we feed the texts into an unsupervised machine

[90]This is a very important phenomenon that requires deeper interpretations. Just imagine web pages of the Internet, whereby every page is directly or indirectly connected to every other page on the network. We know that this is not true in the case of the Internet.

learning and recreate the texts, it will come out like what we have shown above.[91] What it means is that the word with the highest probability is "allah" followed by "lord" and so on. This is probably why the challenge to come out with just one Surah, even a short Surah, is unmet until today.[92] You must have the word "allah" or convey the meaning of oneness or Tauheed, which is impossible for the non-believers.

7.2.6 Clustering coefficients

There are many ways to compute clustering coefficients in a network, we will use the *igraph* method called *transitivity()*. The measure for the network transitivity is at 10.7%. This is a measure of the probability that given a node, the adjacent nodes (words) are connected. The number obtained here is extremely high. For most other real networks, the probabilities are extremely small; for example, the internet (0.003298%), World Wide Web (0.001412%), and E.coli metabolism (0.537055%). An implication of this finding is an indication of how "dense" the words are in the text, and almost no word is left without relations to other words.

For reference, we have seen examples of these clusterings earlier in Chapter 5 for Surah Yusuf and Chapter 6 for Surah Taa Haa.

7.2.7 Modularity

The modularity algorithm is used to find *community structures* or groupings of nodes and edges in large networks. In *igraph*, this is accomplished by applying the *cluster_walktrap()* function. However, this approach has some shortcomings, mainly because it relies on a random walk approach in finding communities, which is sensitive to the starting position and is used mainly in undirected graphs. For this purpose, we rely instead on the "modularity class" function of Gephi for calculations. The results are shown in Figure 179.

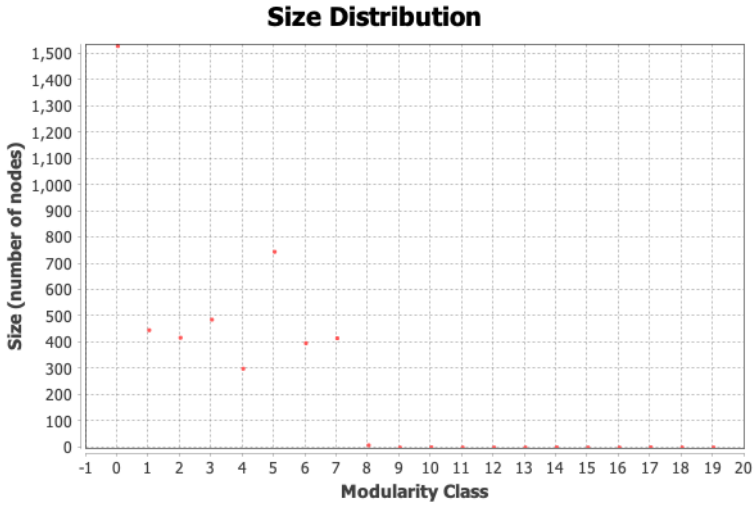


Figure 179: Modularity class

It is interesting to note that there are seven major modular classes with members of 300 or more, with the largest community having about 1,500 members.[93] In fact, the smaller classes are with members of less than ten, and can be ignored (classes of 8 and above). The percentage of nodes within each class is as follows:

[91] Tests using unsupervised learning LSTM Neural Network model was done by the first author; the results of which are not fully ready for publication at the time of this writing.

[92] Refer to verse 2:23

[93] Modularity algorithm is dependent on its setting of resolution limits, which determines how small the communities we want to detect. In our case here we set it to 1, which is the standard limit.

33.63% (one-third of the nodes), 17.87%, 15.68%, 9.58%, 8.33%, 7.42%, and 6.33% (from the first to the seventh).

In grouping terms, we can say that each modular class represents a certain "commonality"; if we want to understand what these commonalities are, then we have to dive deeper into each class and investigate inside the classes. Furthermore, we can set the resolution limit of the modularity measure, which allows us to break the classes into a more refined set. We leave this issue as a future research direction.

In this book, we will just show the visualizations of these classes (or groupings) to demonstrate the forms of shapes of the groups.

Figure 180 provides the total picture of the modularity classes within the network.

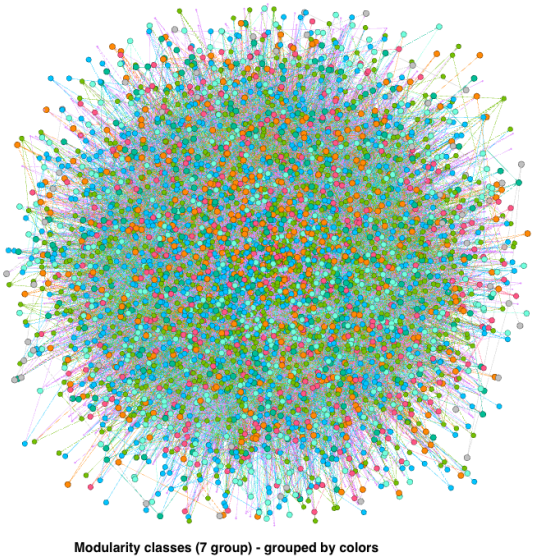


Figure 180: Modularity class by colors

Now let us check the structure of each of the various sub-groups. The largest grouping is shown in Figure 181, which has the same center as the entire network surrounded by words in the same modularity class. Figure 182 shows the second largest group, which has the same center as before but surrounded by another set of words. Figure 183 shows the third largest group is which has the same center as before but surrounded by another set of words.

We can move on to the fourth, fifth, and until the smallest grouping. The key point is what can we learn from these groupings?

First, we want to observe how do the sub-networks look once we take them out of the main network. Do the sub-networks look and behave the same as the main network? Does the main network change when we take out a sub-network? All these questions relate to what is called the "scale-free" properties of a network. A scale-free network is resilient to changes within the network; when a clique is taken out, the clique's properties are the same as the main network properties; while the properties of the network minus the clique also remain the same.

In another word, the network is structured in such a way that it behaves like a fractal - the large part is the combination of many small parts, while the small parts can exist by themselves, co-exist with other small parts, can be combined together and create a larger part, and so on. Fractal properties are resilient to "cascading failures" which is evident in most of nature's physical properties.[94]

We can take the meaning of fractal properties into a much deeper context; for example, are the messages within the sub-network part of the composition of the main network? What happens to the messages in the

[94]For more on fractal properties, please refer to https://en.wikipedia.org/wiki/Fractal.

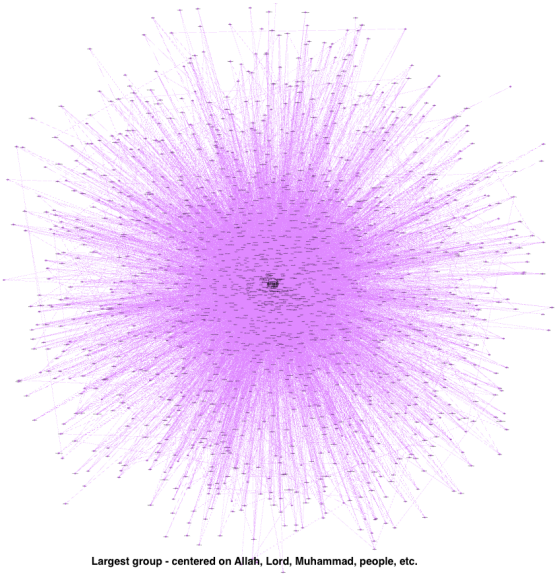


Figure 181: Network of largest clique

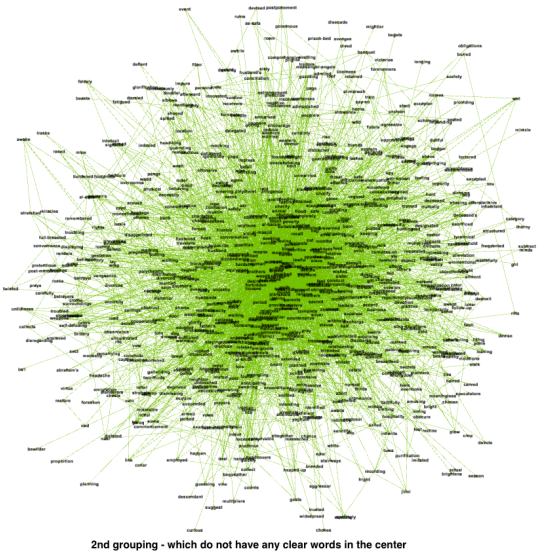


Figure 182: Network of second largest clique

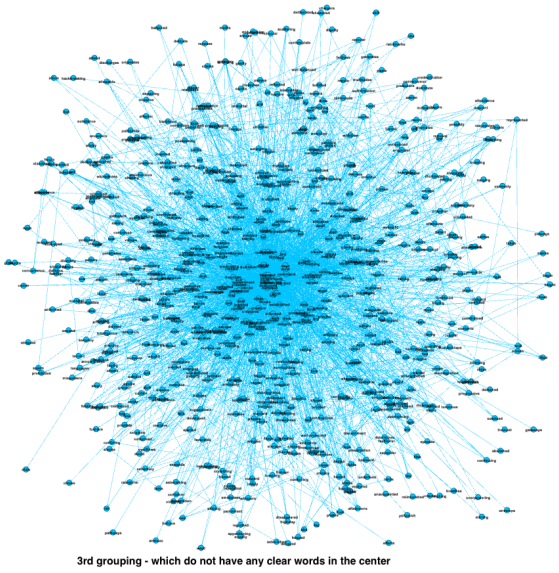


Figure 183: Network of third largest clique

main network when we take out a clique? Can we combine messages in two cliques and are the messages still coherent?

As an example, we can compile all the words within a clique and do a *sentiment analysis* on the subset of words, similar to what we have shown in Chapter 3. We can also weigh the sentiment scores against the position of the word within the sub-network, etc. Whatever meanings that come out is subject to interpretation in terms of what they represent.

There are numerous ways to expand the current analysis, which is beyond the current introductory scope of this book. We will leave it for future work of Quran Analytics. For our purpose here, based on visual observations of the Figures 180, 181, 182, and 183, we can say that the network demonstrates some forms of "scale-free" (and fractal-like) properties. This is an important observation and provides leads for future exploration and analysis.

7.2.8 Betweenness

Betweenness measures the relative importance of words in connecting other words as the word in between. We compute the measures using the following codes:

```
btwnQ = betweenness(igphQ, v = V(igphQ), directed = TRUE,
                    weights = NULL, normalized = FALSE)
top_btwn = btwnQ[rev(order(btwnQ))]
```

Now let us use the results and rewrite the phrase as we have done earlier, using *betweenness* instead, as follows:

allah, lord, people, day, earth

punishment, fear, fire, believed, muhammad

created, evil, disbelievers, truth, messenger

women , moses , bring , hearts , surely

A comparison with words from the top degree, reveals some interesting observations. Except for the top few words, some other words changed their positions. This reflects the meaning of betweenness measures, that is some words appear more as a word in between two words, and lesser in terms of links.

7.2.9 Prestige centrality

Centrality measures refer to the centrality position of a word in the whole text. There are many ways to measure centrality, the simplest one being **eigenvector centrality**. This is the measure of the "importance" of a word. This is computed as the codes below:

```
evcentQ = eigen_centrality(igphQ)
top_evcent =evcentQ$vector
top_evcent = top_evcent[rev(order(top_evcent))]
```

We repeat the same exercise using the "prestige" as the rankings instead.

allah ,people ,lord ,earth ,fear

messenger ,day ,heavens ,knowing ,believed

punishment ,truth ,muhammad ,believers ,wills

merciful ,exalted ,mercy ,worship ,belongs

We do not intend to use the exercise here to interpret Al-Quran using the techniques used. However, for those who studied Al-Quran, some of the things shown here will start to make some sense of the direction of "knowledge" which can be extracted from using these measures of the network. So far, we have dealt only with the top-level words and have not dived deeper to lower-ranking texts (below 20), which might reveal many further insights. Again, we have to leave this for future work.

7.2.10 Summary

Summarizing all the statistical properties of the nodes of the network, we can say that all the important words (top features) have a high degree, betweenness, and prestige centrality within the network. The consistencies of these measures across these top features are very interesting in the sense that the first word, *Allah* is the topmost in all cases (the highest degree, highest prestige, the most betweenness, and also highest in all measures of centrality).

7.3 Dive into selected Surahs

The methods introduced in the previous section can be applied to a Surah as well as in performing comparisons between Surahs. Let us choose two intermediate-length Surahs, namely Al-Kahf (No. 18, 110 verses) and Maryam (No. 19, 98 verses). There are a few approaches we could take:

a) understanding the total network of the texts (tokens) in the Surah;
b) understanding the "topics" of the Surah.

First, we will plot the network and obtain summary statistics.

```
kahf = quran_en_sahih[grep("Al-Kahf",quran_en_sahih$surah_title_en),]
kahf_toks <- kahf$text %>%
    tokens(remove_punct = TRUE) %>%
    tokens_tolower() %>%
    tokens_remove(pattern = stop_words$word, padding = FALSE)
dfm_kahf = dfm(kahf_toks)
fcmKahf <- fcm(kahf_toks,
               context = "window",
               tri = FALSE)
```

```
igphKahf = quanteda.textplots::as.igraph(fcmKahf)
degKahf = degree(igphKahf, v = V(igphKahf),
                 mode = "total", loops = TRUE,
                 normalized = FALSE)
top_degreeKahf = degKahf[rev(order(degKahf))]
btwnKahf = betweenness(igphKahf, v=V(igphKahf), directed = TRUE,
                       weights = NULL, normalized = FALSE)
top_btwnKahf = btwnKahf[rev(order(btwnKahf))]
evcentKahf = eigen_centrality(igphKahf)
top_evcent =evcentKahf$vector
top_evcentKahf = top_evcent[rev(order(top_evcent))]
featKahf <- names(topfeatures(fcmKahf, 50))
```

```
fcm_select(fcmKahf, pattern = featKahf) %>%
    textplot_network(min_freq = 0.5,
                     edge_color = "steelblue",
                     edge_alpha = 0.5,
                     edge_size = 2,
                     vertex.size = 1)
```

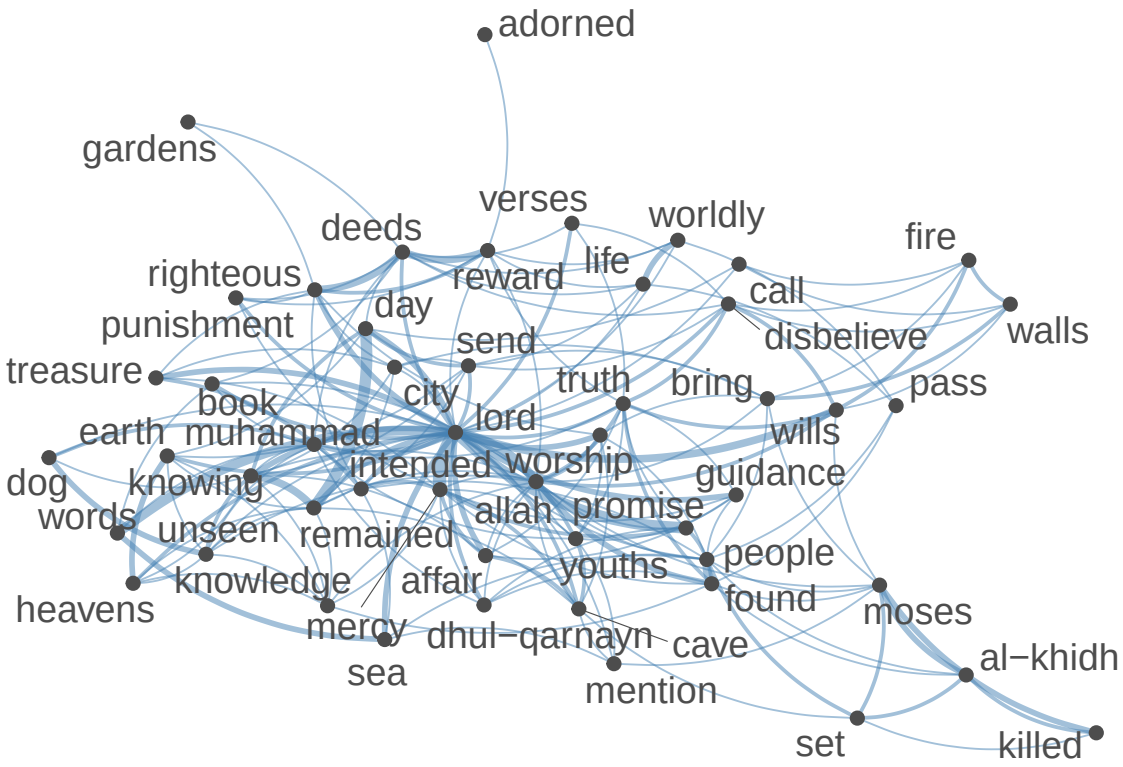


Figure 184: Network of top 50 words in Surah Al-Kahf

```
maryam = quran_en_sahih[grep("Maryam",quran_en_sahih$surah_title_en),]
maryam_toks <- maryam$text %>%
    tokens(remove_punct = TRUE) %>%
    tokens_tolower() %>%
    tokens_remove(pattern = stop_words$word,
```

```
                  padding = FALSE)
dfm_maryam = dfm(maryam_toks)
fcmMaryam <- fcm(maryam_toks,
             context = "window",
             tri = FALSE)
igphMaryam = quanteda.textplots::as.igraph(fcmMaryam)
degMaryam = degree(igphMaryam, v = V(igphMaryam),
                   mode = "total", loops = TRUE,
                   normalized = FALSE)
top_degreeMaryam = degMaryam[rev(order(degMaryam))]
btwnMaryam = betweenness(igphMaryam, v=V(igphMaryam), directed = TRUE,
                         weights = NULL, normalized = FALSE)
top_btwnMaryam = btwnMaryam[rev(order(btwnMaryam))]
evcentMaryam = eigen_centrality(igphMaryam)
top_evcent =evcentMaryam$vector
top_evcentMaryam = top_evcent[rev(order(top_evcent))]

featMaryam <- names(topfeatures(fcmMaryam, 50))
```

```
fcm_select(fcmMaryam, pattern = featMaryam) %>%
    textplot_network(min_freq = 0.5,
                     edge_color = "tomato",
                     edge_alpha = 0.5,
                     edge_size = 2,
                     vertex.size = 1)
```

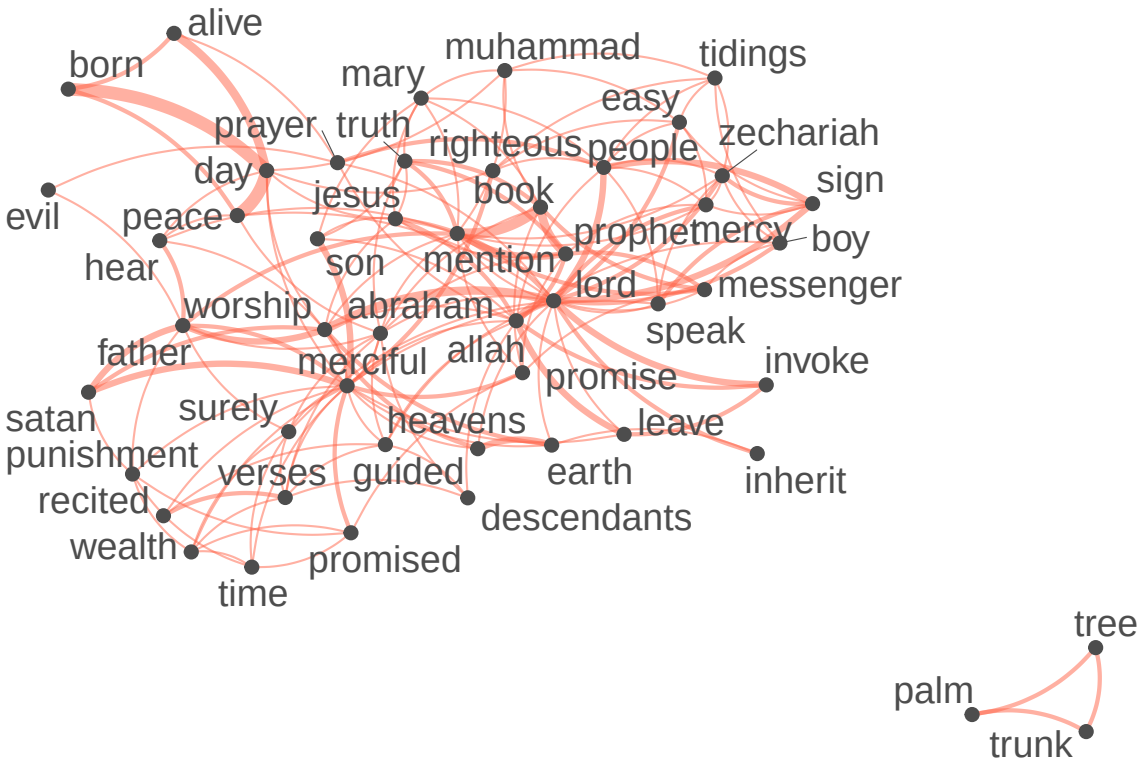


Figure 185: Network of top 50 words in Surah Maryam

For both Surahs, as shown in the figures, the most prominent word at the center is “lord”; however the

surrounding topics are different. In Al-Kahf words such as “cave”, “youth”, “al-khidh”, “moses” emerge, while in Surah Maryam “merciful”, “jesus” appear.

The summary statistics are tabulated in the Table:

Statistics	Surah Al-Kahf	Surah Maryam
Nodes	506	326
Edges	5,516	2,838
Average degree	21.8	17.41
Diameter	7	7
Average Path Length	3.06	3.24
Size of largest component	504	315
Transitivity	0.28	0.33

The table above reveals that the network characteristics of both Surahs are not far from each other. Despite Al-Kahf being a bit longer, which results in more nodes and edges, the diameter, average distance to the center, are all about the same.

Now let us try to get some ideas regarding the thematic subjects of both Surahs, from the three elements of measures: degree, clustering, and betweenness.

Surah Al-Kahf	Text
By degree	lord ,allah ,people ,moses ,found ,earth ,mercy deeds ,day ,remained ,bring ,truth ,cave ,knowing righteous ,reward ,sea ,muhammad ,al-khidh ,send
By betweenness	lord ,allah ,moses ,people ,earth ,reward ,day bring ,gardens ,deeds ,cave ,beneath ,remained ,found dog ,muhammad ,send ,affair ,truth ,time
By prestige centrality	lord ,allah ,mercy ,promise ,remained ,words ,knowing day ,wills ,righteous ,affair ,truth ,deeds ,exhausted worship ,cave ,muhammad ,sea ,associate ,guidance

One way to interpret the keywords is that the main keywords are (in the order of): “lord”, “allah”, then by the number of mentions: “people”, by between the subjects: “moses”, and by the prestige of mention: “mercy”.

Surah Maryam	Text
By degree	lord ,merciful ,allah ,day ,mention ,father ,people worship ,prophet ,surely ,abraham ,book ,prayer ,boy jesus ,verses ,punishment ,peace ,promised ,earth
By betweenness	lord ,merciful ,allah ,day ,died ,alive ,surely father ,people ,worship ,prayer ,abraham ,hear ,brought mention ,earth ,jesus ,speak ,angel ,destroyed
By prestige centrality	lord ,allah ,worship ,people ,merciful ,day ,mention sign ,invoke ,prophet ,jesus ,father ,leave ,book unhappy ,zechariah ,mercy ,boy ,peace ,abraham

One way to interpret the keywords is that the main keywords are (in the order of): “lord”, then by the number of mentions: “merciful”, by between the subjects: “merciful”, and by the prestige of mention: “worship”.

7.4 Word collocations statistical method

In this section we will take another approach to the computations of word collocations, namely using various statistical tools. Most of these tools utilize what is called "distance measures", which mathematically is the calculation relative statistical scoring of multi-word expressions. The multi-word can be set to two (i.e. bigrams) or three (i.e. trigrams), and so on, depending on the objective of the analysis.

The function we can apply is *textstat_collocations()* from *quanteda*. Here we show how this function is applied and what the results meant(Blaheta and Johnson, 2001).

What is the most important two-word sequence in Surah Al-Kahf?

```
tscKahf = kahf_toks %>% quanteda.textstats::textstat_collocations(method = "lambda", size = 2,
                        min_count = 1,smoothing = 0.5,tolower = TRUE)
tscKahf[tscKahf$count == max(tscKahf$count),]$collocation
```

What is the most important three-word sequence in Surah Al_Kahf?

```
tscKahf = kahf_toks %>%
              textstat_collocations(method = "lambda", size = 3,
                        min_count = 1,smoothing = 0.5,tolower = TRUE)
tscKahf[tscKahf$count == max(tscKahf$count),]$collocation
```

What is the most important two-word sequence in Surah Maryam?

```
tscMaryam = maryam_toks %>%
              textstat_collocations(method = "lambda", size = 2,
                        min_count = 1,smoothing = 0.5,tolower = TRUE)
tscMaryam[tscMaryam$count == max(tscMaryam$count),]$collocation
```

What is the most important three-word sequence in Surah Maryam?

```
tscMaryam = maryam_toks %>%
              textstat_collocations(method = "lambda", size = 3,
                        min_count = 1,smoothing = 0.5,tolower = TRUE)
tscMaryam[tscMaryam$count == max(tscMaryam$count),]$collocation
```

The results in the exercises indicate that out of all possible collocations (bigrams for two-word phrases and trigrams for three-word phrases), these resulting phrases rank highest in the entire text (i.e. the chosen Surah). They stood out as the most "outstanding phrases" which explain the subject of the texts (i.e. Surah). Whether the results make any meaningful sense or not is a subject of the interpretation of the texts (i.e., translated text of Al-Quran).

As a comparison, we will do the same for Surah Maryam for Yusuf Ali (instead of Saheeh as done before). Will we get the same answer? (We will show only the results since the codes follow the same steps).

```
quranEY <- quran_en_yusufali %>%
             select(surah_id,
                    ayah_id,
                    surah_title_en,
                    surah_title_en_trans,
                    revelation_type,
                    text,
                    ayah_title)
maryamEY = quranEY[grep("Maryam",quranEY$surah_title_en),]
maryamEY_toks <- maryamEY$text %>%
    tokens(remove_punct = TRUE) %>%
    tokens_tolower() %>%
    tokens_remove(pattern = stop_words$word,
                  padding = FALSE)
```

What are the most important two-word sequences in Surah Maryam in Yusuf Ali?

```
tscMaryamEY = maryamEY_toks %>%
                textstat_collocations(method = "lambda", size = 2,
                    min_count = 1,smoothing = 0.5,tolower = TRUE)
tscMaryamEY[tscMaryamEY$count == max(tscMaryamEY$count),]$collocation
```

What are the most important three-word sequences in Surah Maryam in Yusuf Ali?

```
tscMaryamEY = maryamEY_toks %>%
                textstat_collocations(method = "lambda", size = 3,
                        min_count = 1,smoothing = 0.5,tolower = TRUE)
tscMaryamEY[tscMaryamEY$count == max(tscMaryamEY$count),]$collocation
```

We can see that Yusuf Ali's translations will give a different phrase. In fact, if we go deeper into phrases of slightly lower ranking, some of the phrases do match.

7.5 Word keyness comparisons

If we want to compare the keywords of one Surah against another Surah, we can use the *keyness* statistical measures. In *quanteda* we can use *textstat_keyness()* function and also *textplot_keyness()* for plotting the results. Let us compare prominent keywords in Surah Al-Kahf versus Maryam. This is shown in Figure 186.

```
kahf_maryam = rbind(kahf,maryam)
corp <- corpus(kahf_maryam$text)
docvars(corp, "surah") <- c(rep("kahf",110),rep("maryam",98))
dfmat2 <- dfm(corp, remove = stop_words$word, dfm_group = "surah",
              remove_punct = TRUE)
tstat2 <- textstat_keyness(dfmat2, measure = "lr") # target = "kahf", measure = "lr")
textplot_keyness(tstat2, color = c("steelblue", "tomato"), n = 20)
```

The keyness plot shows and confirms what is known about the two Surahs. In Surah Al-Kahf, the word "found" relates to the cave-dwellers and al-khidh (Khidir) as well as Dhul-Qarnayn. In Surah Maryam, the key message is "merciful", an attribute of Allah, and Maryam (Mary) and her son Isa (AS) as signs of His mercy.

7.6 Lexical diversity and dispersion

Lexical diversity is a measure of how each sentence adds to the lexical variety in terms of its addition to the vocabulary within a corpus. Let us make a comparison between Surah Al-Kahf and Surah Maryam, by calculating *textstat_lexdiv()* scores and plotting the results side by side.

```
kahf_lexdiv = textstat_lexdiv(dfm_kahf, measure = "all")
maryam_lexdiv = textstat_lexdiv(dfm_maryam, measure = "all" )
p1 = ggplot() + geom_point(aes(x = 1:nrow(kahf_lexdiv),
                               y = kahf_lexdiv$R),
                               color = "steelblue",
                           show.legend = FALSE) +
                labs(x = "Surah Al-Kahf verses", y = "lexical diversity score")
p2 = ggplot() + geom_point(aes(x = 1:nrow(maryam_lexdiv),
                               y = maryam_lexdiv$R),
                               color = "tomato",
                               show.legend = FALSE) +
```

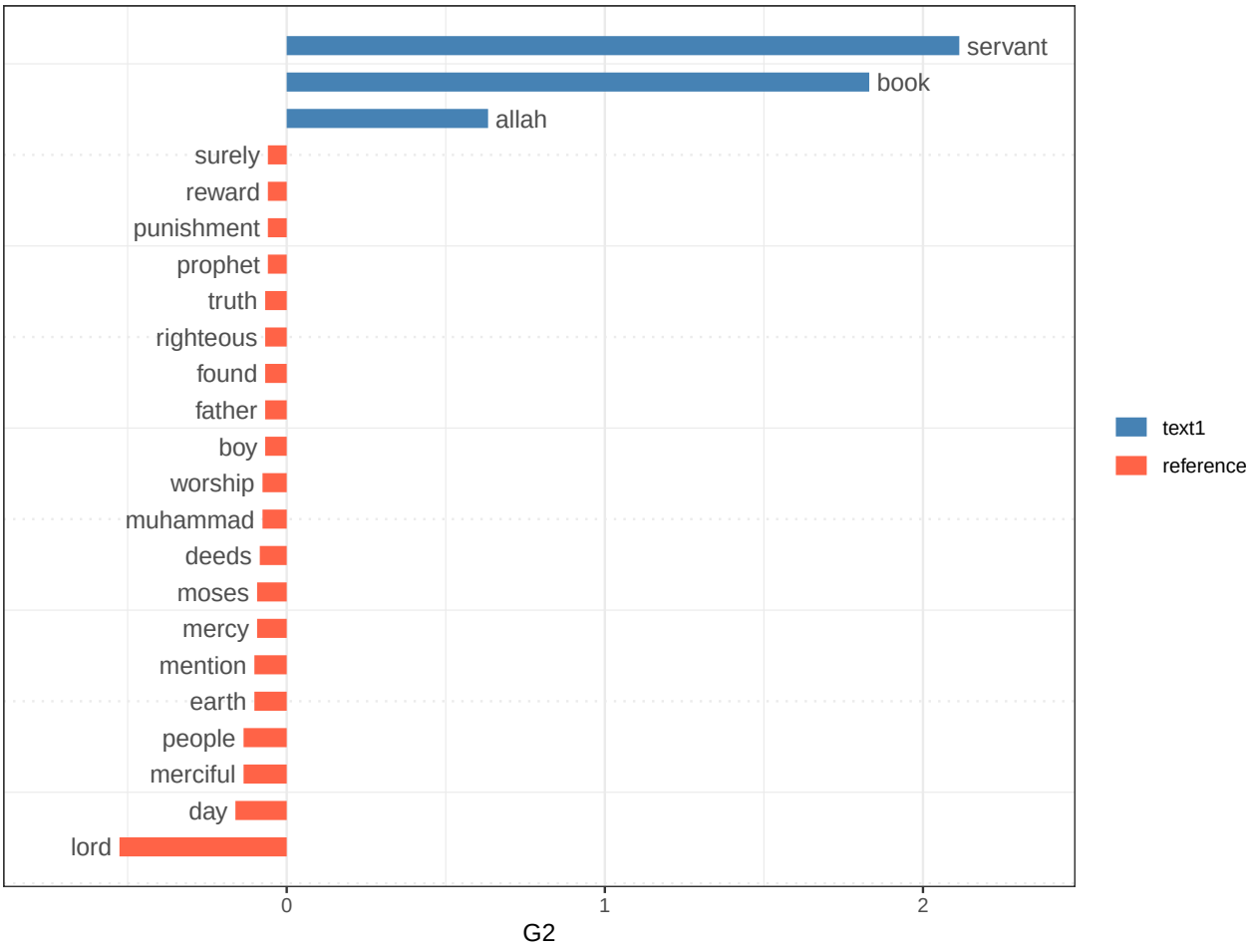


Figure 186: Keyness plot for Surah Maryam and Al-Kahf

```
                labs(x = "Surah Maryam verses", y = "lexical diversity score")
cowplot::plot_grid(p1,p2, nrow = 1)
```

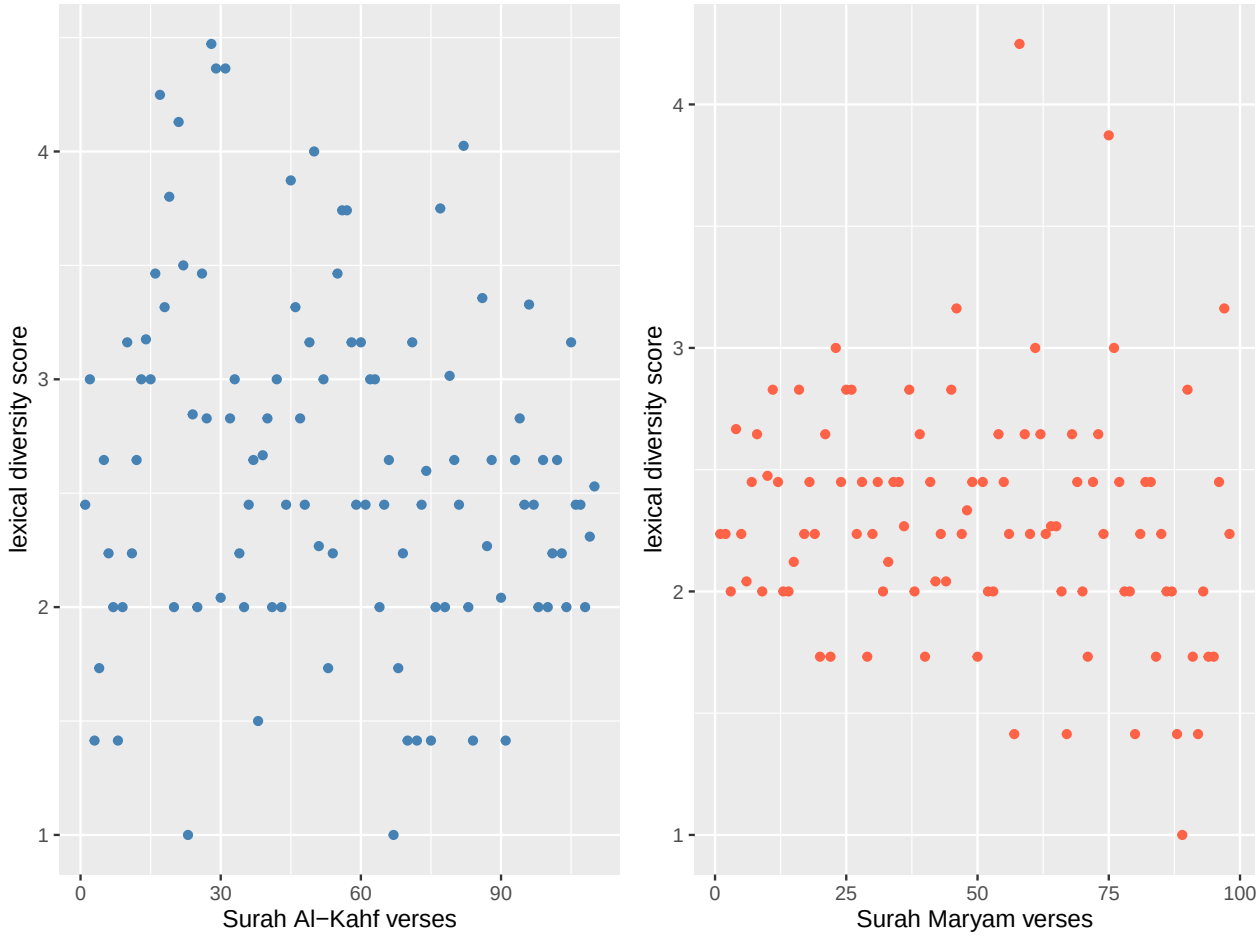


Figure 187: Lexical diversity scores for Surah Al-Kahf and Maryam

We can observe from Figure 187 that the verses in Surah Al-Kahf are much more diverse in their lexical diversity, throughout the Surah; whereas Surah Maryam's later verses (after verse 50) show more variety. What this implies is that the vocabulary structure in the verses of Surah Al-Kahf is different from Surah Maryam.

Another approach of comparison is called *key words in context* or *kwic* lexical dispersion plot. Note that "text1" to "text110" is from Surah Al-Kahf and the rest ("text111 to"text208") are from Surah Maryam.

First, let us apply it to the word "Allah" and "Lord".

```
textplot_xray(kwic(corp, pattern = "allah"), kwic(corp, pattern = "lord"),
              scale = "absolute") +
              aes(color = keyword) +
              scale_color_manual(values = c("darkblue", "darkred")) +
              theme(legend.position = "none")
```

The plots in Figure 188 display the frequency of the selected keyword and its appearance within the various verses ("texts"). Lexical dispersion demonstrates the richness of emphasis of the whole document regarding the message, via frequencies of occurrence relative to the sentences (verses) within the document.

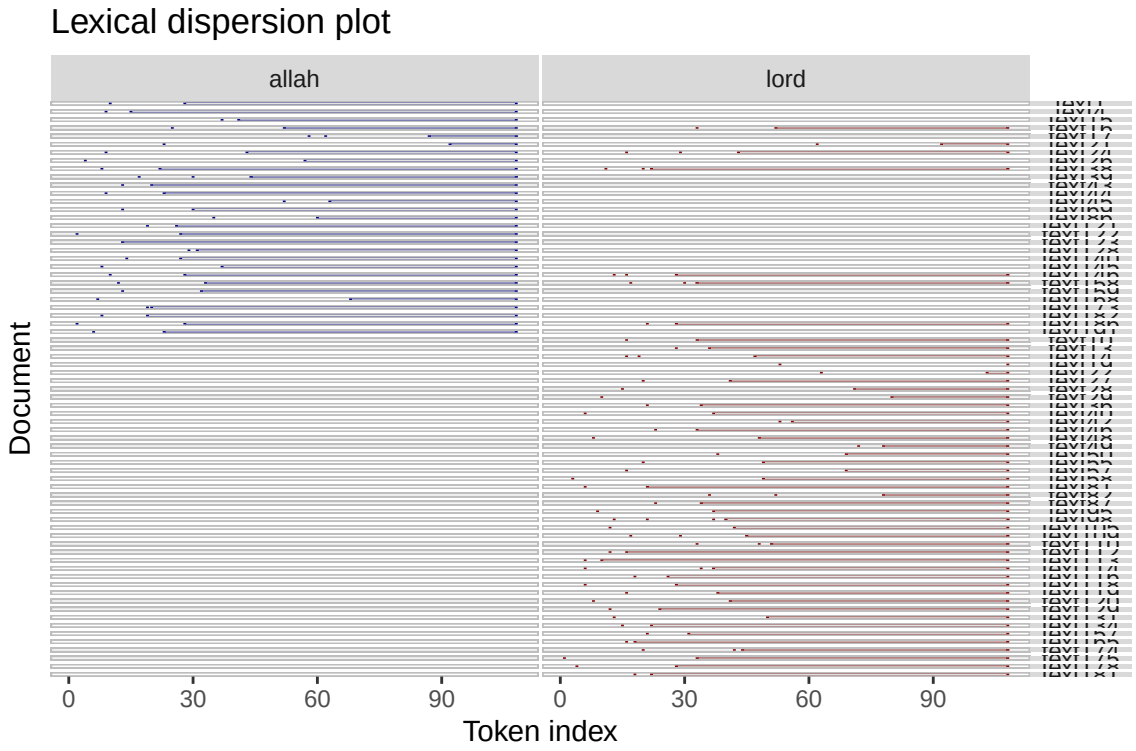


Figure 188: Keyword in context (kwic) plot for Surah Maryam and Al-Kahf for the word 'Allah' and 'Lord'

The word "Allah" appears lesser than "Lord" in both Surahs. There are some occasions when "Allah" is mentioned, "Lord" is also mentioned (i.e. the lines where both blue and red dots exist). However, there are many verses in which "Lord" is mentioned without the mention of the word "Allah" (lower parts of the plot).

What the exercise does is to get some sense into why, lexically, certain words appear in some verses (or Surahs), and do not appear in some other verses (or Surahs). The explanation of why there are patterns of appearance is a subject of the interpretation of Al-Quran. The method we use here is only a tool to detect and visualize the patterns.

7.7 Viewing the network as dendrogram

We would like to end by presenting another tool that is useful for viewing a large network of co-occurrence data as we have been dealing with in the previous sections. The dendrogram is useful to view "clustering" or "grouping" of the networks using algorithms such as *fast_greedy* or *cluster_edge_betweenness* as discussed in Chapter 5.

Let us show the results for both Surahs.

We can see from Figure 189 and Figure 190 that there are clusterings or groupings in the cooccurrences words; in fact there are about three large groupings in Surah Al-Kahf and Surah Maryam. We can extract out the data for the groupings and do further analysis as needed and required. We leave the subject as it is for our work here.

7.8 Words similarity in verses

The concept of similarity in texts in NLP applies to sentences or in our case verses. How similar are any two verses in a set of verses, such as a Surah? Or how similar are verses from one Surah compared to verses

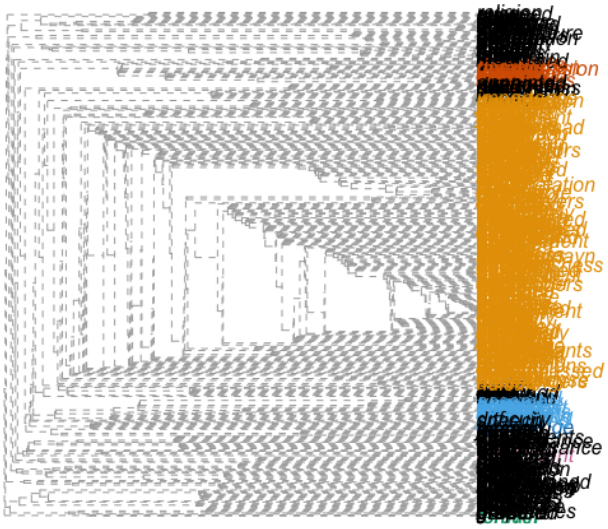


Figure 189: Dendrogram for clusters in Surah Al-Kahf

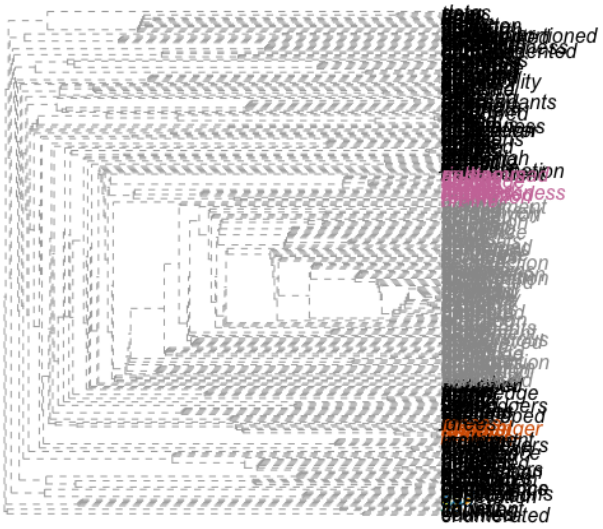


Figure 190: Dendrogram for clusters in Surah Maryam

from another Surah? This is like posing a question: does a verse contains similar words (not necessarily in the same order) with another given verse?

In linear algebra, the measure is the dot product of two vectors of not necessarily similar length, normalized to the product of their Euclidean norms, called *cosine similarity* ($cosine_{sim}(x, y) = \frac{x.y}{|x|.|y|}$). Similar measure is called *jaccard similarity*, which is the measure of set similarity instead of dot product ($jaccard_{sim}(a, b) = \frac{|a \cap b}{|a \cup b|}$).

This can be applied using the *textstat_simil()* function in *quanteda*.

```
tstcor_kahf = textstat_simil(dfm_kahf, y = NULL,
               margin = "documents",
               method = "correlation")
tstcor_kahf = as.data.frame(tstcor_kahf)
tstcor_kahf = tstcor_kahf %>% filter(correlation > 0)
tstcos_kahf = textstat_simil(dfm_kahf, y = NULL,
               margin = "documents",
               method = "cosine")
tstcos_kahf = as.data.frame(tstcos_kahf)
tstjac_kahf = textstat_simil(dfm_kahf, y = NULL,
               margin = "documents",
               method = "jaccard")
tstjac_kahf = as.data.frame(tstjac_kahf)
ggplot() +
  geom_point(aes(x = c(1:943), y = tstcor_kahf$correlation), color = "red") +
  geom_point(aes(x = c(1:943), y = tstcos_kahf$cosine), color = "blue") +
  geom_point(aes(x = c(1:943), y = tstjac_kahf$jaccard), color = "green") +
  labs(x = "edges", y = "similarity score")
```

Figure 191 shows the plot for three measures namely *correlation* (red), *cosine* (blue), *jaccard* (green). Two verses are 100% similar textually, word-for-word, and a good number with a similarity score above 25% (i.e. more than a quarter of the words). What do all these numbers mean depends on how interpretations are made; for example, it can be said that these verses explain each other.

```
tstcos_kahf$document1 = str_replace(tstcos_kahf$document1,"text","")
tstcos_kahf$document2 = str_replace(tstcos_kahf$document2,"text","")
igph_kahfcos = graph_from_data_frame(tstcos_kahf %>%
                                     filter(cosine > 0.25))

ggraph(igph_kahfcos, layout = "kk") +
  geom_edge_link(aes(width = cosine, edge_alpha = cosine),
                 edge_color = "steelblue") +
  geom_node_point(size = 0.1, shape = 1, color = "black") +
  geom_node_text(aes(label = name), col = "black", size = 4)
```

The best way is to view them as an *igraph* plot using *ggraph* (as discussed in the previous chapter). Figure 192 shows clusters of verses that are linked. If we check further, the verses around v65 to v78, are repeated conversations between Moses and Khidh. The other cluster surrounding v38, is about the story of the cave dwellers. What we have shown is how statistical tools in NLP are used to find related sentences (or verses) in a large text (if we apply them to the entire translation).

For completeness, we will show similar plots for Surah Maryam (Figures 193 and 194). We leave the readers to check why the verses are linked in this manner for the Surah.

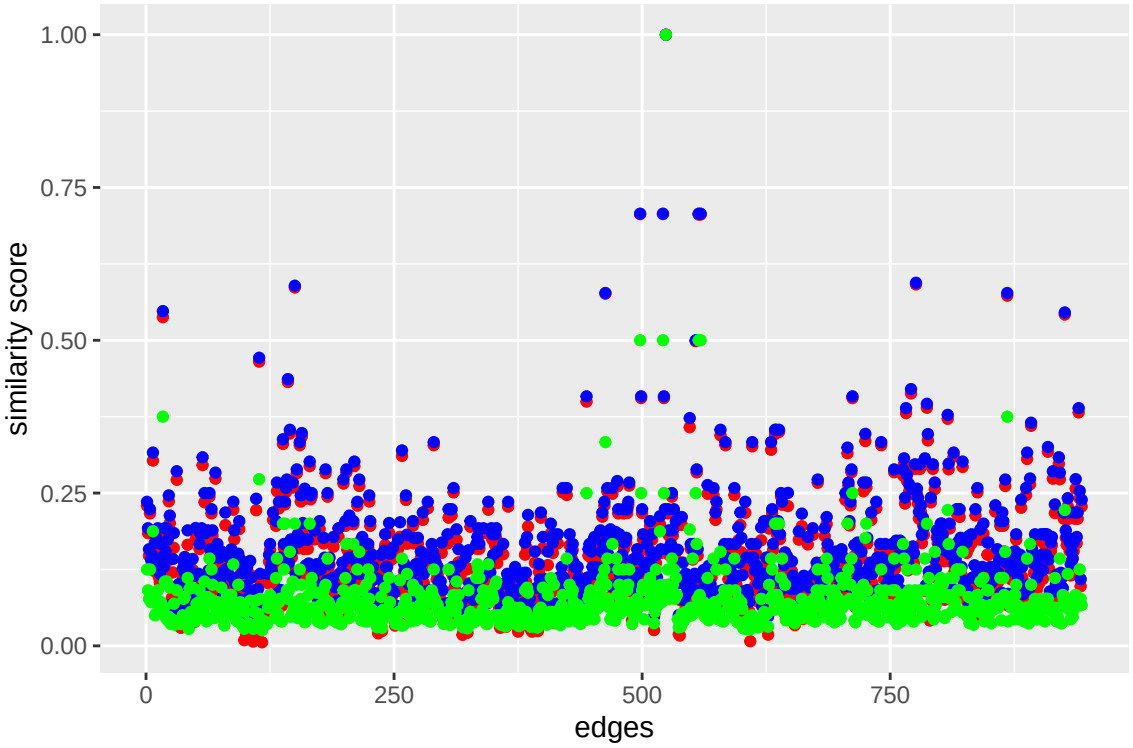


Figure 191: Verses similarity for Surah Al-Kahf using correlation, cosine and jaccard measures

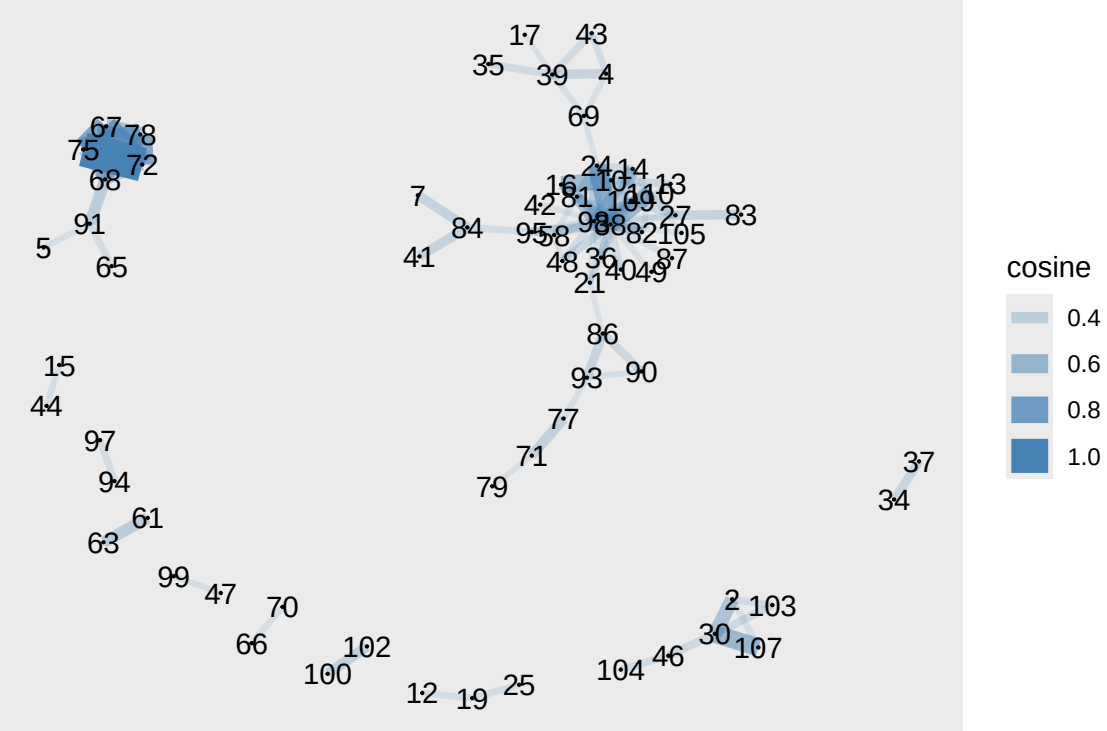


Figure 192: Verses with high similarities for Surah Al-Kahf

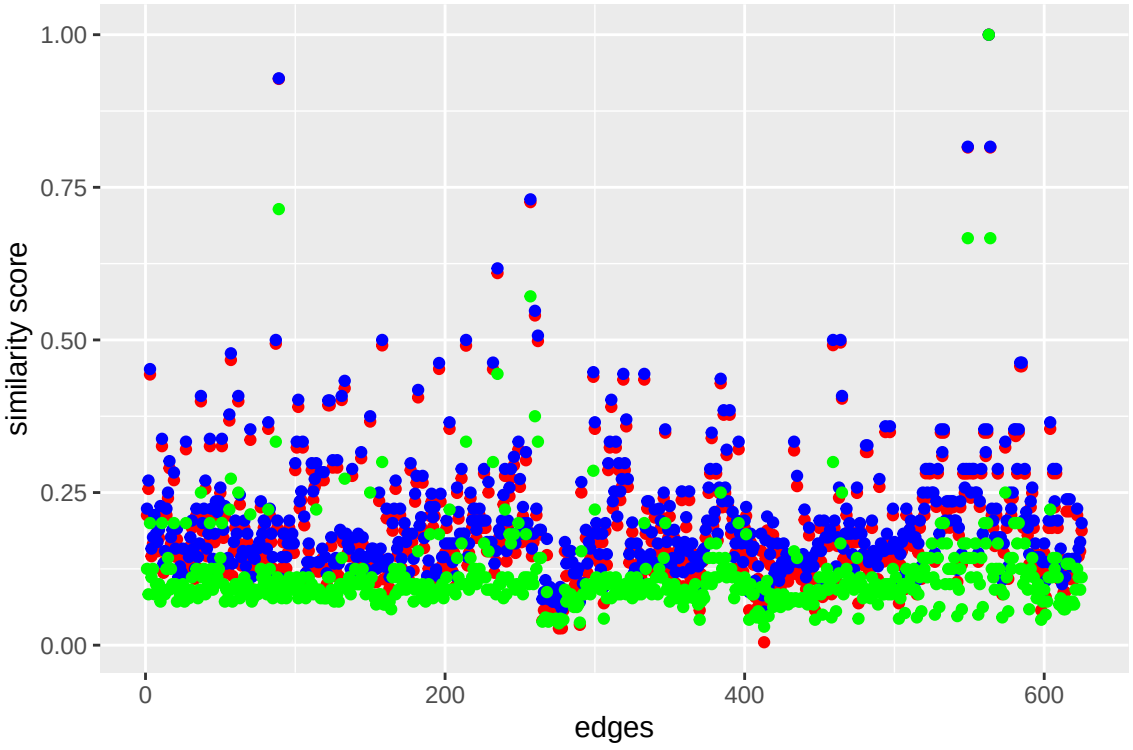


Figure 193: Verses similarity for Surah Maryam using correlation, cosine and jaccard measures

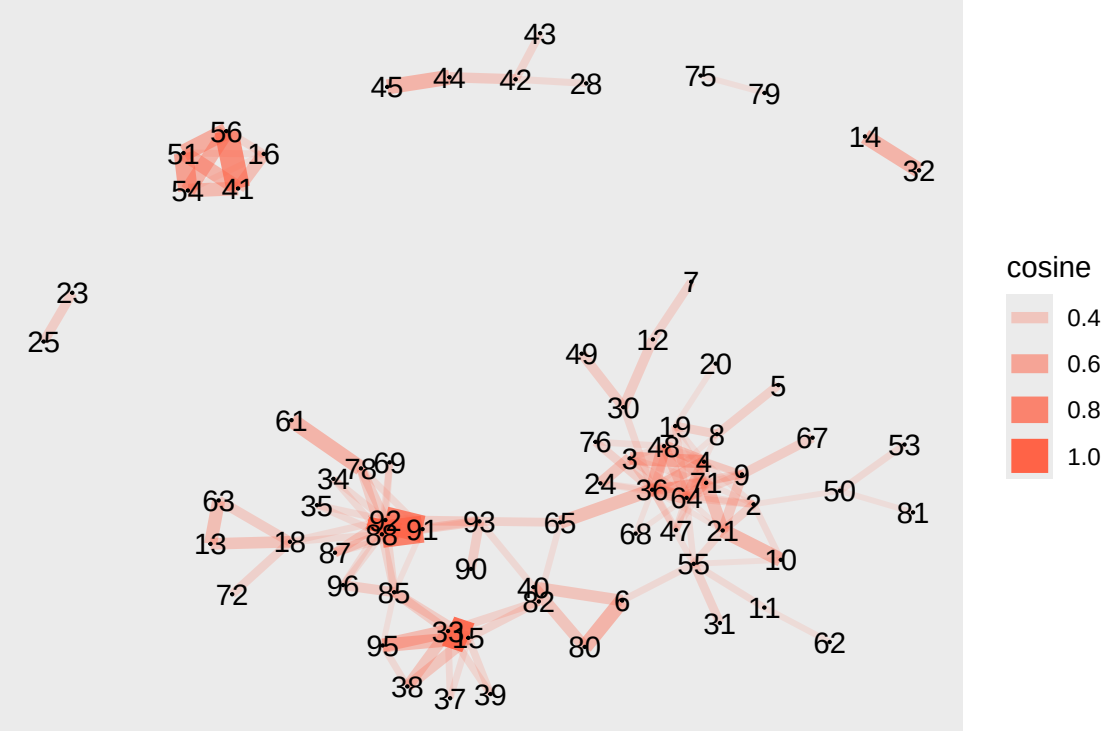


Figure 194: Verses with high similarities for Surah Maryam

7.9 Words dissimilarity in verses

The inverse of the concept of similarity is the concept of distance in text. How different are any two verses in a set of verses, such as in a Surah will be?

In linear algebra, the measure is the simple Euclidean distance of two vectors.[95] There are few options in *textstat_dist()* function in *quanteda* besides *euclidean*, like *minkowski*[96] and *manhattan*[97]. We will use *manhattan*, because it is the most amplified version of distance.

```
tstdis_kahf = textstat_dist(dfm_kahf, y = NULL,
               margin = "documents",
               method = "manhattan")
tstdis_kahf = as.data.frame(tstdis_kahf)
ggplot() + geom_point(
              aes(x = c(1:nrow(tstdis_kahf)),
                  y = scale(tstdis_kahf$manhattan, scale = F)),
              color = "aquamarine2") +
              labs(x = "edges", y = "distance score")
```

Figure 195: Verses distance for Surah Al-Kahf using manhattan measures

Figures 195 and 196 guide us to check, why verse 45 is very different than verse 29. Many similar exercises are possible by extracting out the data (as plotted) and analyze them in whichever ways a researcher deems suitable.

The methods of similarity and distance are different from sentiment scoring in Chapter 3. Here the focus is on similar words existing in two different sentences. The more the similarity is, the higher the cosine (or

[95] https://en.wikipedia.org/wiki/Euclidean_distance
[96] https://en.wikipedia.org/wiki/Minkowski_distance
[97] https://xlinux.nist.gov/dads/HTML/manhattanDistance.html

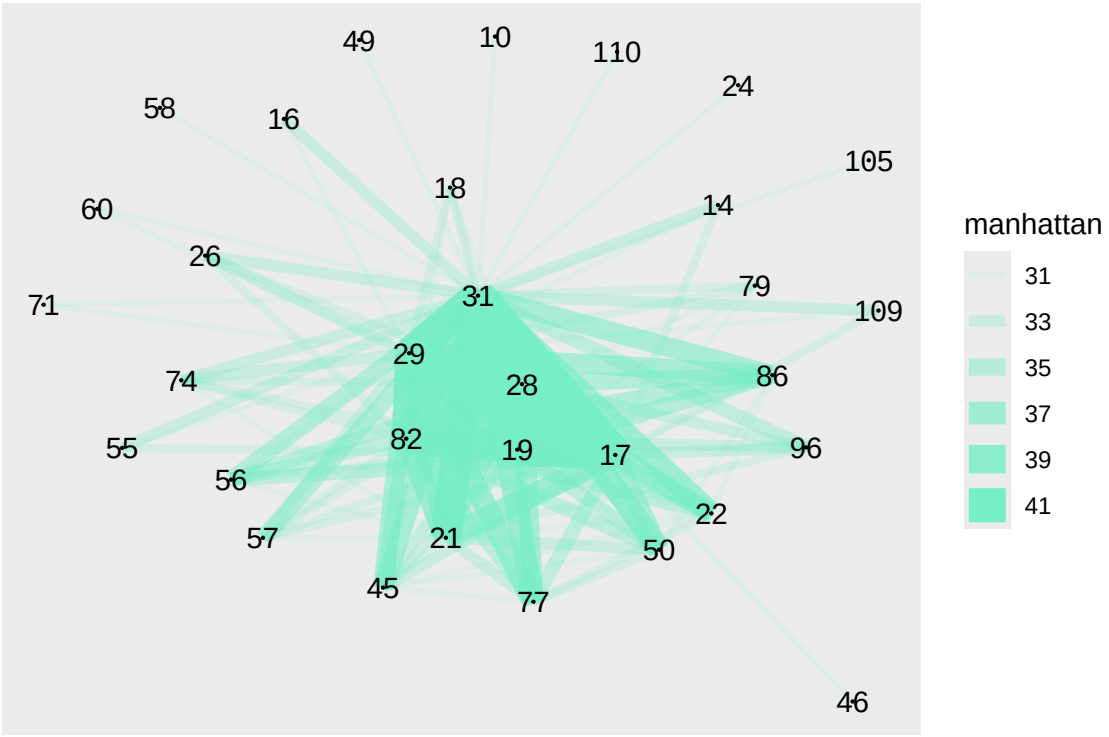


Figure 196: Verses with high distances for Surah Al-Kahf

other similarity measures) is. If there is no word matching, then the higher the distance is (measured by euclidean or manhattan scores).

We will not repeat the exercises for Surah Maryam and we also will not do similar exercises on Yusuf Ali for brevity. We leave it for readers' own exercise using the **R** codes enclosed together with the book.

7.10 Summary

We have demonstrated using the *quanteda* package how to work with various NLP tasks for the Saheeh English translation of Al-Quran. The applications can be extended to include all other versions of English translations and also non-English translations using the methods shown. One particular point to emphasize is, we have used tools that are language-model independent. None of the methods shown require any assumptions of pre-built models of a language (except for the use of pre-built stopwords). These are non-parametric methods that we promoted as the basis for unsupervised learning methods, which is a fast-developing area in the applications for NLP tasks.

We have provided only glimpses of what are the possible tools and methods, which a researcher of Quran Analytics can take further. We provide some suggestions here.

1. Expand the usage of the tools to develop methods of analysis based on the objective of linguistics and analytical studies for translations of Al-Quran and benchmark it against linguistic models (for the selected language) and against the original text of Al-Quran.
2. Network models of texts are versatile and expandable in many directions, as a non-parametric approach and application of graph theory or network science algorithms.
3. Language, which consists of words as one of its basic elements, can be viewed as a system or systems. Network models of language allow us to study language from the perspective of network dynamics and systems theory.

4. Tools of NLP in **R** such as *quanteda* are extremely versatile because they do not force us to make prior underlying assumptions. We can just let the statistical results and visuals open up interesting questions that we can explore further.

7.11 Further readings

quanteda package documentation (https://quanteda.io).

quanteda manual and guides (Benoit et al., 2018)

Ban, K., Meštrović, A., and Martincic-Ipsic, S. (2014). *Initial comparison of linguistic networks measures for parallel texts* (Ban et al., 2014).

8 Text Classification Models

The NLP task for modeling classifications of words for complex subjects such as modeling content analysis of ideas, opinions, and sentiments from texts or speeches is difficult due to a few factors. Too little data renders the exercise prone to large errors but too big of data infuse much noise (or entropy) which confounds the models' measurement. Too much data with insufficient entropy causes the overfitting of models. There is no clear start and also there are no clear ends.

For example, as we have shown in Chapter 3, sentiment scoring is clearly model-dependent; the results vary if we vary the model used. Whether a pre-built model is a good scoring method for Quran Analytics is yet to be ascertained.

This chapter serves as an introduction to the subject of NLP text modeling, focused on a very specific model, "topic modeling". This is among the easiest of the models involved. Expanding the task to other higher and more complex dimensions is beyond the scope of this book, as our intent is to introduce the subject and demonstrate some of the tools for Quran Analytics.

We will introduce many tools for the task of NLP text modeling, which are: *quanteda.textmodels* (Benoit et al., 2020) package - which is an extension of *quanteda* as we have seen in Chapter 7. We will also introduce Structural Topic Model *stm* package (Roberts et al., 2020), *topicmodels* (Grün et al., 2020), and *text2vec* (Selivanov et al., 2020) package which is a similar wrapper to the famous Stanford NLP Group's GloVe: Global Vectors for Word Representation,[98] which is a word embedding tool, an extremely versatile tool for Machine Learning tasks in NLP.

8.1 Brief outline of text modeling

In this section, we present a brief outline of the task of modeling classifications and the position of each method within the larger NLP tasks framework. The summary provided here is from (Grimmer and Stewart, 2013).

The tasks start with defining the research objective, which either involves ideological scaling or general classification. Under both areas, the methods will be divided into either supervised or unsupervised; and within each, there will be various possible methods of applications depending on whether the problem involves "known categories" (or labeling) or "unknown categories" (unknown labels).

Labeling (or annotation) is a tedious manual process if it is humanly done, and humans are prone to errors and biased judgements. Automated labeling on the other hand is prone to algorithmic errors, which may remain undetected until later. Both methods require validations, and continuous updating and validations. However, with the advancement of computing and algorithms, the task is better left to computers than humans; except for extremely skewed cases if they come into consideration.

[98] Please see https://nlp.stanford.edu/projects/glove/ for details.

Furthermore, all models involve statistical inferencing and causality analysis. As we have shown in Chapter 2, the distributional properties of word frequencies follow fat-tail or non-gaussian statistical distributions, specifically, it follows the Power Law structure. The presence of these properties in the data causes many other issues within inferencing, namely errors in both; modeling errors and errors in the model. Hence, due care is required in dealing with any models which are based on the standard assumptions.

The chart in Figure 197 provides a full scenario of the possible paths of modeling.[99]

8.1.1 General setting

Most standard text models are a form of generative probabilistic model, which requires a statistical estimation process based on optimization of a cost function. The objective of the process is to obtain a set of estimates (or estimators) that optimizes the cost (i.e. lowest cost) for a predefined loss function. The variations between a type of model to another are in:

a) the choice of the loss function,
b) the method of estimation,
c) the type of estimators, and
d) the statistical properties of the estimators.[100]

The formal setting for the models are as follows:[101]

- a *word* is the basic unit of discrete data, indexed by a number for each word in the vocabulary derived for the entire dataset, as $V = (1, 2, ..., K)$. This is the basic category in the data.
- a *document* is an ordered sequence of N *word*, $w = (w_1, w_2, ..., w_N)$ where w are numbers in V
- a *corpus* is a collection of ordered M *document*, consisting of ordered *word*, denoted by $d = (d_1, d_2, ..., d_M)$

The first step for pre-processing the data involves converting the entire corpus into *data*. The steps are: a) tokenizing, b) create a vocabulary for the tokens present, c) create the metadata for the word, document, and corpus.

All these steps were explained in previous chapters using pre-built functions in *tidytext* or *quanteda*.

The texts, now in data format, are converted to vector representations, using the vocabulary V as its look-up table (for converting back into textual form). Notationally they are as follows:

$$C = set(D, W)$$

A form of representation of C is the Document Term Matrix (DTM) as we have seen before, and the Feature Co-occurrence Matrix (FCM) is a reduced compact form of C based on co-occurrence representation.

In statistical terms, what we have now are observations, w_i, which are the individual *word*, and its measures are the location (observation) and its frequencies. At a higher level, the observations are d_j and its measures are the numerical representation of it, which is its score. Given the setting, then the statistical problem is set as follows:

$$F_{argmin}(\theta) = LOSS(X)$$

[99] This is our own remake of points from (Grimmer and Stewart, 2013) paper.
[100] The basic setting is based on standard or canonical statistical inference methods.
[101] The notations here follow (Blei et al., 2003).

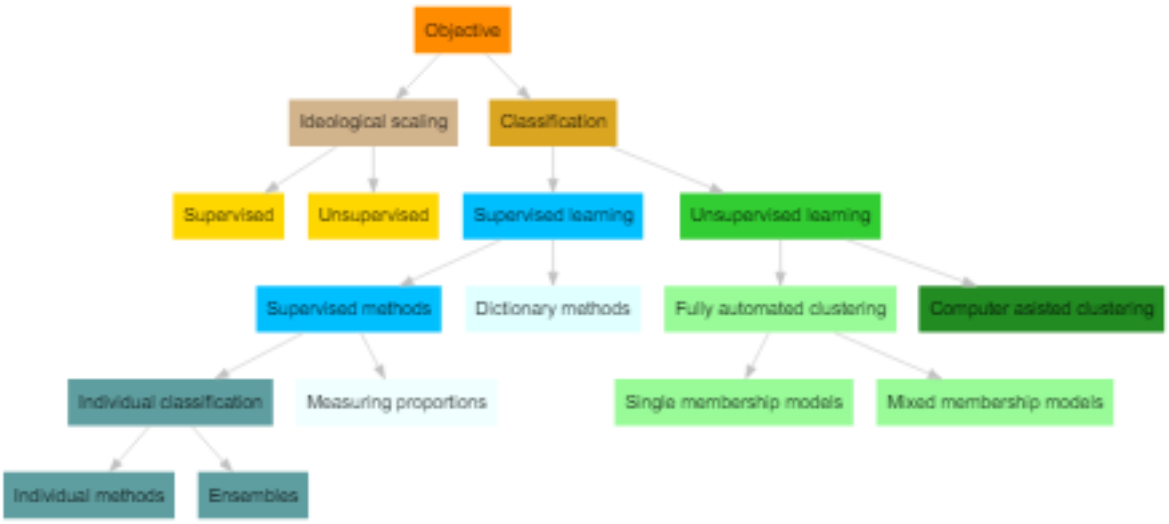


Figure 197: Text modeling framework in NLP

We want to find an estimate of θ which minimizes the loss function. Once we obtain the estimate, called $\hat{\theta}$, we can use it to infer and obtain the estimation. The estimation (or inference) will be numbers of probability (i.e. between 0 and 1), which are the "probability scores" based on the model.

Different text models will use different model assumptions and loss functions. We provide a concise explanation for a few choices of text models that we will use in this book.

8.1.2 Supervised and unsupervised learning methods

The major difference between the supervised and unsupervised learning method is in whether the model assumes known categories (also called labels) or unknown categories (labels) of the texts under study. We can generally categorize supervised as pre-built language models, such as the one applied in the sentiment scoring example (in Chapter 3). Unsupervised models, on the other hand, utilize a generalized process of clustering, either using an in-built process of clustering (called fully automated clustering) or in-process clustering (using some algorithms) as part of the pre-processing of the data (called computer-assisted clustering).

The latest development of the supervised versus unsupervised models involves infusion of both types into each other, which falls under the name of "headless AI", "Unsupervised AI". The process involves combining the steps of unsupervised learning as the start, then using the results to auto-generate the labels, which are then implied into supervised learning models.[102] The main difference of the infused model is the process of labeling is entrusted to the algorithms rather than human.

Since human language is a complex subject, the performance of any models, supervised, unsupervised, or both combined is still a subject that requires in-depth studies and testing. The encouraging part is, as we study more data as they become available, by way of big data, with better and faster computing power, our ability to deal with language through computational linguistics has improved tremendously.

8.1.3 Topic modeling for Quran Analytics

In this book, we will explore a specific task in NLP called "topic modeling". This is an unsupervised machine learning method, suitable for the exploration of textual data. The calculation of topic models aims to determine the proportionate composition of a fixed number of topics in the documents of a collection. Since it is an unsupervised machine learning method, it is useful to experiment with different parameters in order to find the most suitable parameters for our own analysis needs.

Topic Modeling often involves the following steps.[103]

1. Read in and preprocess text data,
2. Calculate a topic model using a text model (using unsupervised learning model),
3. Visualize the results from the calculated model and
4. Select documents based on their topic composition.

Topic modeling is an interesting aspect of NLP in the sense that, given a large amount of text data, which structures are not known offhand, and we want to extract information about the text, particularly what are the "headlines" or "themes" of the text in question. The general idea is, if a term is present in many of the sentences, in a certain particular structure, then we say that it has a high probability to be the key subject of the discussion. It implies that the word has a high probability to be in the headlines or themes, in the same way as to how we think a news headline should be. Putting it another way, given the text of news without the headlines, can we guess what the headline should be?

For Quran Analytics, we know Al-Quran is arranged in a very deterministic manner (i.e. fixed permanently), and no change has occurred to the original text. Particularly each word, from the first to the last, is arranged

[102] An example of this approach is h2o.ai, https://www.h2o.ai
[103] https://tm4ss.github.io/docs/Tutorial_6_Topic_Models.html

in verses, from the first to the last (in 6,236 verses), in Surahs, from the first to the last (in 114 Surahs). Verses represent themes (or messages) of their own, and Surahs also represent a theme or combination of themes. The themes may be repeated in various words, or verses, or various parts within the Surahs, as well as various parts of the entire Al-Quran.

To perform a thorough and complete task of uncovering themes from Al-Quran, based on translated texts, is no small feat. It should be a research subject by itself. Since the purpose of this book is to introduce the subject, we will scale it down to demonstrations of the various models and focus on a specific segment of Al-Quran, through a selected Surah, namely Surah Al-Kahf.

8.2 Unsupervised learning models

As explained earlier, unsupervised learning models assume that we do not know off-hand the labeling of the texts, such as the topics of the texts under study. Our task is to "uncover" those labels, which have a high probability to be the contents of the topics for the collection of texts.

In this section we will cover some of the well known unsupervised learning models, namely:

1. Latent Dirichlet Allocation (LDA)
2. Latent Semantic Analysis (LSA)
3. Structural topic models (STM)

First, we will provide a quick technical brief of the models.

8.2.1 Latent Dirichlet Allocation (LDA) model[104]

LDA is a special class factorial multinomial probit model, conditioned on topic z_l, where z_l is drawn from a Poisson distribution; and the loss function is defined as $Dir(\alpha)$ as $p(w_n|z_l, \beta)$, a multinomial probability over the space of β. The Dirichlet function Dir is the defining function of the model. The function relies on a method of sampling, which is pre-determined as part of the modeling process. In most applications, the standard method is "Gibbs" sampling.[105]

The performance of LDA has shown promising results in various applications, in particular for topic modeling on some challenging aspects, such as in information extraction from text records, social media sentiment or aspect mining, and others.[106]

In our work here, we will use the *topicmodels* package, which contains the *LDA()* function.

8.2.2 Structural Topic Models (STM)

The foundation of STM is provided by (Roberts et al., 2014) and developed as an **R** package, *stm* (Roberts et al., 2020), details of which are provided in its package article (Roberts et al., 2019) and its webpage guide.[107]

STM uses a Generalized Linear Model (GLM) framework by incorporating document meta-data into the topic modeling process. The process follows the same approach of LDA, where it is conditioned on topic z_l. z_l is drawn from a Logistic Normal; and the loss function is defined as $p(w_{d,n}|z_{d,l}, \beta_{d,k=z_{d,n}})$, a multinomial probability over the space of β. The structure is the same, but with different assumptions of conditionality and probability distributions of LDA.

In our work here, we will use the *stm* package, which contains many neat functions for visualization of the results.

[104] For a more detailed explanation, please refer to (Blei et al., 2003).
[105] For a reference on Gibbs sampling, please see: https://en.wikipedia.org/wiki/Gibbs_sampling
[106] A comprehensive survey of LDA model applications is provided by (Jelodar et al., 2019).
[107] https://www.structuraltopicmodel.com

8.2.3 Latent Semantic Analysis model

LSA was first introduced by (Landauer et al., 1998) and has been around a bit longer than LDA and STM. The objective of LSA is to replicate the human cognitive phenomena using a statistical learning model. It assumes that word-for-word, word-sentence (or passage), sentence(or passage)-for-sentence (or passage) relations are correlated in the way of association or semantic similarity, within textual data.

The statistical process of LSA involves applying Single Value Decomposition (SVD) to the text matrix data (such as a DFM or DTM); from which it generates a new orthogonal space of factor values (based on the dimensions, which are the number of topics - a dimension reduction process). In the current time, the process has a resemblance to a Neural Network (for those who are familiar with it) - which is a non-parametric method of performing simultaneous multiple regression equations, and the final results will be a correlation matrix for the topics and the texts (and sentences).[108]

In our work here, we will use the *quanteda.textmodels* function called *textmodel_lsa()*.

8.2.4 Labeling the data

We will use Surah Al-Kahf as our text under study. We choose the Surah for a particular reason; it is a medium length Surah, containing six rather distinct sections:

1) the story of the cave dwellers (v1-v31),
2) the story of two garden owners (v32-v44),
3) a parable of worldly life (v45-v59),
4) the story of Prophet Moses and Al-Khidh (v60-82),
5) the story of Dhul-Qarnayn (v83-102), and
6) about deeds (v103-v110).

Based on the observation, we will label the verses according to our own interpretation and investigate the subject as we go along, first using unsupervised learning, then supervised learning. In unsupervised learning, we will use the labels as a check between the model against our assumption; and in the supervised model, we will use the model to check its accuracy, against what we assume as the actual situation.

```
quran_all = read_csv("data/quran_trans.csv")
kahf <- quran_all %>% filter(surah_title_en == "Al-Kahf") %>%
                mutate(Group = ifelse(ayah %in% 1:31, "cave",
                                      ifelse(ayah %in% 32:44, "garden",
                                      ifelse(ayah %in% 45:59, "life",
                                      ifelse(ayah %in% 60:82, "moses",
                                      ifelse(ayah %in% 83:102, "dhul",
                                      "deeds"))))))
dfm_kahf <- kahf$saheeh %>%
    tokens(remove_punct = TRUE) %>%
    tokens_tolower() %>%
    tokens_remove(pattern = stop_words$word, padding = FALSE) %>% dfm()
```

8.2.5 Latent Dirichlet Allocation (LDA)

We will start with the most common algorithm for topic modeling, namely *Latent Dirichlet Allocation (LDA)*. LDA is guided by two principles.

1. Every document is a mixture of topics. We imagine that each document may contain words from several topics in particular proportions. For example, in a two-topic model, we could say "Document 1 is 80% topic A and 20% topic B, while Document 2 is 40% topic A and 60% topic B."

[108] For detail exposition, please refer to (Landauer et al., 1998).

2. Every topic is a mixture of words. Importantly, words can be shared between topics; a word like "deeds" might appear in both equally.

LDA is a mathematical method for estimating both of these at the same time: finding the mixture of words that are associated with each topic, while also determining the mixture of topics that describes each document.

For parameterized models such as LDA, the number of topics k is the most important parameter to define in advance. How an optimal k should be selected depends on various factors. If k is too small, the collection is divided into a few very general semantic contexts. If k is too large, the collection is divided into too many topics of which some may overlap and others are hardly interpretable.

For our analysis, we choose a thematic "resolution" of $k = 6$ topics. In contrast to a resolution of 100 or more, 6 topics can be evaluated qualitatively very easily. We also set the seed for the random number generator to ensure reproducible results between repeated model inferences.

We use the *LDA()* function from the *topicmodels* (Grün et al., 2020) package, setting $k = 6$, to create a six-topic LDA model. Almost any topic model in practice will use a larger k, but we will soon see that this analysis approach extends to a larger number of topics. The function returns an object containing the full details of the model fit, such as how words are associated with topics and how topics are associated with documents.

We need to "clean" the Document Feature Matrix, *dfm* from empty rows and run the *LDA()* function as shown below:

```
require(topicmodels)
drop <- NULL
for(i in 1:NROW(dfm_kahf)){
     count.non.zero <- sum(dfm_kahf[i,]!=0, na.rm=TRUE)
     drop <- c(drop, count.non.zero < 1)
   }
dfm_kahf_clean <- dfm_kahf[!drop == TRUE,]
k <- 6
lda_kahf <- topicmodels::LDA(dfm_kahf_clean, k, method="Gibbs",
            control=list(iter = 300, seed = 1234, verbose = 25))
```

Depending on the size of the vocabulary, the collection size, and the number k, the inference of topic models can take a long time. This calculation may take several minutes. If it takes too long, reduce the vocabulary in the DTM by increasing the minimum frequency in the previous step.

The topic model inference results in two (approximate) posterior probability distributions: a distribution theta over k topics within each document and a distribution β over V terms within each topic, where V represents the length of the vocabulary of the Surah (V = 240).

We take a look at the 7 most likely terms within the term probabilities β of the inferred topics.

```
library(tidytext)
kahf_topics <- tidy(lda_kahf, matrix = "beta")
kahf_top_terms <- kahf_topics %>%
  group_by(topic) %>%
  top_n(7, beta) %>%
  ungroup() %>%
  arrange(topic, -beta)

kahf_top_terms %>%
  mutate(term = reorder_within(term, beta, topic)) %>%
  ggplot(aes(beta, term, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
```

```
  facet_wrap(~ topic, scales = "free") +
  scale_y_reordered()
```

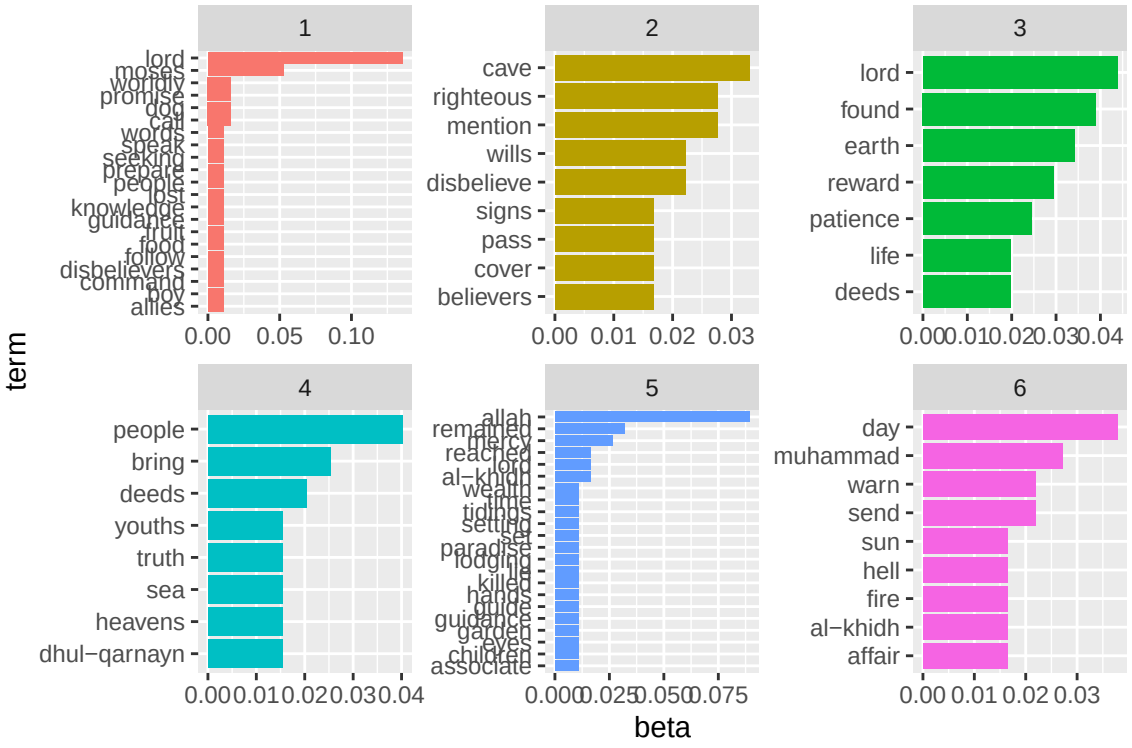


Figure 198: Top terms with LDA methods for Surah Al-Kahf

From Figure 198 we can see that the first topic contains the term "moses" and the second topic contains "cave", and so on along the lines of the six topics we assumed. However, we see the various terms are mixed within the topics. As an example, the term "al-khidh" appears in a few of the top terms in two of the topics, and various other terms crisscrossing among the topics.

The reason why terms can be mixed between topics, despite we assuming that it shouldn't happen (for example we know precisely that "al-khidh" appears only in the specific segment), is due to the fact that the model does not "know" the topics off-hand. It only assumes six topics which location within the texts is assumed to be of equal probability across the entire text. What LDA does is to coerce six dimensions onto the data and figure out how best the six dimensions "fit" the data. Whatever the outcome is the result of this process.

Furthermore, LDA tries to generate a pecking order for the topics, by giving one of the dimensions (i.e. topic) the highest rank from any of the six possibilities, followed by the second dimension and so forth. We can check which topics have a "higher" hierarchical clustering by checking its similarity using a distance method (as explained in Chapter 7), such as a Euclidean distance.[109]

```
lda.similarity <- as.data.frame(lda_kahf@beta) %>%
  scale() %>%
  dist(method = "euclidean") %>%
  hclust(method = "ward.D2")
plot(lda.similarity, xlab = "")
```

[109]Mathematically what this means is that we measure the dimensional entropy across the six possible dimensions, and rank the dimension with the highest entropy first, followed by the next.

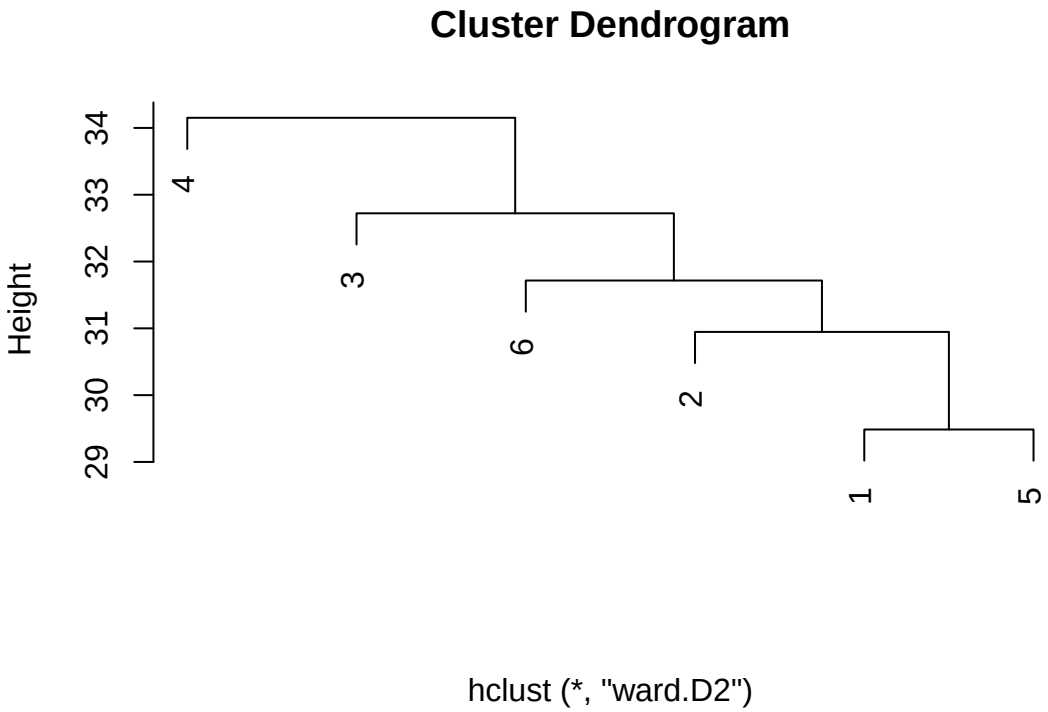


Figure 199: LDA topic similarity by features for Surah Al-Kahf

From Figure 199, we can say that the topic at the highest level is Topic 4, followed by Topic 3, Topic 6, Topic 2, and finally Topic 1 and Topic 5.

Let's take topic 4 as our sample (being the highest rank) and put the terms into a sentence as follows:

> people - bring - deeds - youths - truth - sea - heavens - dhul-qarnayn

This is a typical output of text modeling exercise, where grammatically and semantically the subjects are necessarily clear for a reader. A careful reading of the text shows that the top terms within any topic do not follow exactly what we got from the text (i.e. the Saheeh translation of the Surah). Why this is the case, is a clear example of what has been noted by linguists, as "grammar leaks", which in essence means that grammars are not easily captured within a textual analysis. This issue is particularly notable when we use most of the unsupervised learning methods, such as the LDA.

8.2.6 Structural Topic Models (STM)

Let us look at an alternative method dealing with topic modeling, namely *stm* or Structural Topic Models.

```
library(stm)
dfmkahf_stm <- quanteda::convert(dfm_kahf, to = "stm")
stm_kahf <- stm(
  dfmkahf_stm$documents,
  dfmkahf_stm$vocab,
  K = 6,
  data = dfmkahf_stm$meta,
  init.type = "Spectral"
)
```

We can view the results of the model using the following codes:

```
labelTopics(stm_kahf, c(1:6))
```

We can see from the output that there four different statistics which can be used to rank the terms for a topic; the highest probability, *FREX* (frequency), *Lift*, and *Score*. Different terms will rank higher if we use different measures and the ranking also changes with the measures. What we want to emphasize is that the "curse of dimensionality" is a major issue which we face in these types of analyses.

Let us plot the top words in topics:

```
kahf_topics2 <- tidy(stm_kahf, matrix = "beta")
kahf_top_terms <- kahf_topics2 %>%
  group_by(topic) %>%
  top_n(7, beta) %>%
  ungroup() %>%
  arrange(topic, -beta)

kahf_top_terms %>%
  mutate(term = reorder_within(term, beta, topic)) %>%
  ggplot(aes(beta, term, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  scale_y_reordered()
```

Figure 200: STM topic shares for Surah Al-Kahf

We can see, from Figure 200 that STM brings different results compared to LDA. Which one is more accurate, is impossible to tell from the results. We can produce the same hierarchical plot as we did for the LDA as follows:

```
plot(stm_kahf, type = "summary", text.cex = 1,
  xlab = "Share estimation"
)
```

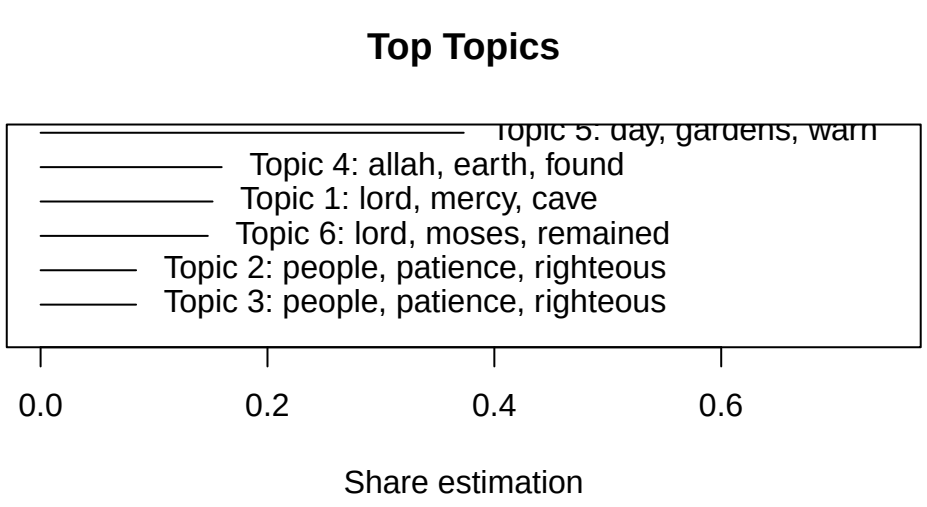


Figure 201: STM topic shares for Surah Al-Kahf

Figure 201 and the results for top-words in Figure 200 demonstrate clearly that STM methods apply a different dimensionality reduction approach than the LDA. From the plot of share estimation in Figure 201, about 45% of the texts are explained by Topic 5, followed by about 16% by Topic 1 and Topic 6.

There are few other neat options within STM that are useful and we will demonstrate them here. One of the functions is *findThoughts()*, which is a tool to find the subjects of a topic within the texts. We will try to get two verses in which Topic 5 and Topic 2 are dominant, as a sample:

```
thoughts4 = findThoughts(stm_kahf, n = 2,text = kahf$saheeh[1:107],
                         topics = 5)$docs[[1]]
thoughts3 = findThoughts(stm_kahf, n = 2,text = kahf$saheeh[1:107],
                         topics = 2)$docs[[1]]
par(mfrow = c(1, 2),mar = c(.5, .5, 1, .5))
plotQuote(thoughts4, width = 45, main = "Topic 5")
plotQuote(thoughts3, width = 45, main = "Topic 2")
```

We present the results in a wordcloud format in Figure 203.

```
cloud(stm_kahf)
```

We can visualize the correlations between the topics in Figure 204:

```
stm_mod_corr = topicCorr(stm_kahf)
plot(stm_mod_corr)
```

Figure 205 is a perspective comparison between the lowest share estimate, Topic 2, against the highest, Topic 5.

```
plot(stm_kahf, type = "perspectives", topics = c(5,2))
```

Topic 5

Those will have gardens of perpetual residence; beneath them rivers will flow. They will be adorned therein with bracelets of gold and will wear green garments of fine silk and brocade, reclining therein on adorned couches. Excellent is the reward, and good is the resting place.

And present to them an example of two men: We granted to one of them two gardens of grapevines, and We bordered them with palm trees and placed between them [fields of] crops.

Topic 2

And nothing has prevented the people from believing when guidance came to them and from asking forgiveness of their Lord except that there [must] befall them the [accustomed] precedent of the former peoples or that the punishment should come [directly] before them.

And [mention] when We said to the angels, "Prostrate to Adam," and they prostrated, except for Iblees. He was of the jinn and departed from the command of his Lord. Then will you take him and his descendants as allies other than Me while they are enemies to you? Wretched it is for the wrongdoers as an exchange.

Figure 202: Sample of verses highly associated with Topic 5 and Topic 2

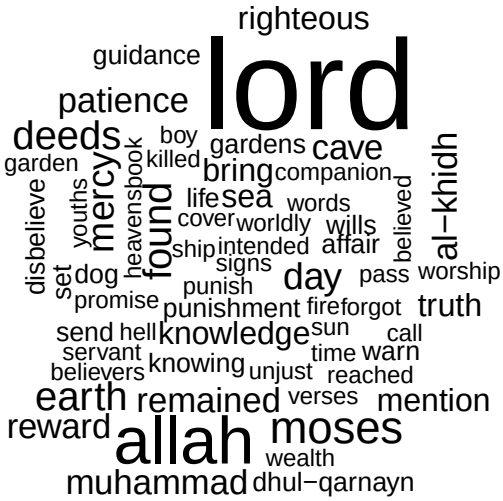


Figure 203: Top words from all topics in wordcloud from STM for Surah Al-Kahf

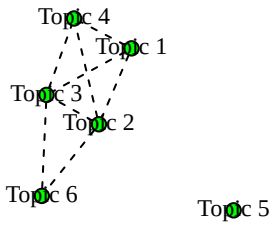


Figure 204: Graphical display of topic correlations from STM for Surah Al-Kahf

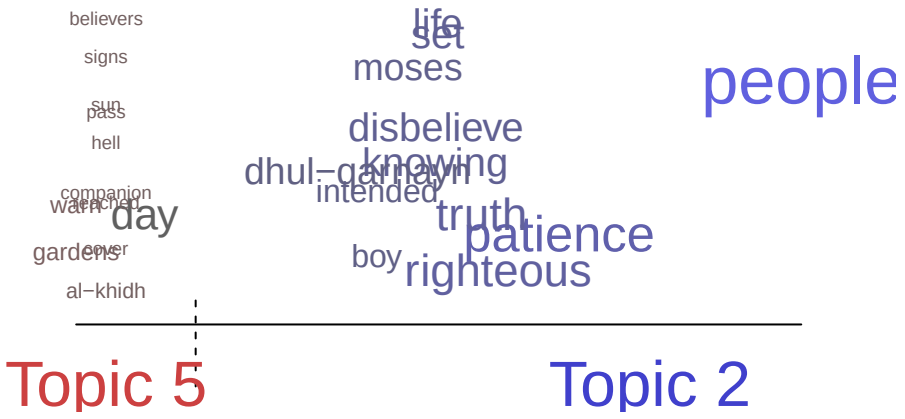


Figure 205: Two perspective for topics from STM in Surah Al-Kahf

Figure 205 shows that all that the computer sees are numbers and their dimensions (i.e. model); where all the texts are represented by probabilities (by the sizes of the texts) and distances (by positions of the texts). It is not easy to convert these numbers and dimensions into a humanly readable format.

The important thing to note is that these models take words in the texts as "they are", without altering their positions (except for stopwords removals) and capture their occurrences. From there on, the models apply statistical calculations based on the formula provided within the models. Statistically speaking, the results are how the data "speak for itself". The exact meanings are for the human to interpret.

The table below provides top-terms to top-terms comparison from both the STM and LDA results, by ranking order.

Topic	STM
Rank 1	(Topic 5) allah, day, muhammad
Rank 2	(Topic 1) lord, earth, mercy
Rank 3	(Topic 6) people, cave, sea
Rank 4	(Topic 4) deeds, found, remained
Rank 5	(Topic 3) moses, righteous, life
Rank 6	(Topic 2) moses, righteous, guidance

Topic	LDA
Rank 1	(Topic 4) people, bring, deeds
Rank 2	(Topic 3) lord, found, earth
Rank 3	(Topic 6) day, muhammad, warn
Rank 4	(Topic 2) cave, righteous, mention
Rank 5	(Topic 1) lord, moses, worldly
Rank 6	(Topic 5) allah, remained, mercy

We can see the impact of text entropy on the results, where the method choice changes the outcome dramatically. Not only does the topic change its ranking, but the terms also change its ranking within a topic and across topics.

8.2.7 Latent Semantic Analysis (LSA)

Latent Semantic Analysis is said to be better at capturing the "semantical" elements within the texts (Landauer et al., 1998). We will see the results and compare them to the STM and LDA methods earlier.

First, let us explain the approach of LSA. There are two different methods of vectorization which we can utilize: vectorization over the features (similar to LDA and STM) or vectorization over the verses. Both have their advantages and disadvantages, depending on our objective. For our purpose, we will do both, first by the features method, so that we can compare the results with STM and LDA. In both methods, we fix the number of topics to be six, by setting $nd = 6$ (six dimensions).

```
lsa_kahf = quanteda.textmodels::textmodel_lsa(dfm_kahf, nd = 6, margin = "both")
```

```
topic1[1:5]
topic2[1:5]
```

LSA works by calculating every individual word within the texts, it computes the score of a sentence (i.e., verses) in the texts, and computes the "distances" between the sentences (verses). These distances are captured through the dimensions of vectorized space of six dimensions (since we chose six topics as our target). The top words for each topic (i.e. dimensions) are the headers as printed above.

Comparing with previous results from STM, LDA:

Topic	STM
Rank 1	(Topic 5) allah, day, muhammad
Rank 2	(Topic 1) lord, earth, mercy
Rank 3	(Topic 6) people, cave, sea
Rank 4	(Topic 4) deeds, found, remained
Rank 5	(Topic 3) moses, righteous, life
Rank 6	(Topic 2) moses, righteous, guidance

Topic	LDA
Rank 1	(Topic 4) people, bring, deeds
Rank 2	(Topic 3) lord, found, earth
Rank 3	(Topic 6) day, muhammad, warn
Rank 4	(Topic 2) cave, righteous, mention
Rank 5	(Topic 1) lord, moses, worldly
Rank 6	(Topic 5) allah, remained, mercy

Topic	LSA
Rank 1	(Topic 1) lord, allah, mercy
Rank 2	(Topic 5) adorned, resting, wills
Rank 3	(Topic 6) wills, truth, disbelieve
Rank 4	(Topic 4) allah, cave, guide
Rank 5	(Topic 3) people, lord, mercy
Rank 6	(Topic 2) lord, words, righteous

To see how this looks, we plot the sentences scoring for each verse (denoted by the number), and across dimensions (we chose dim1 vs dim2, and dim1 vs dim 3, for illustration). Sentences that are "semantically" closer based on "topic a" vs "topic b" are clustered together. This is shown in Figure 206.

```
Name = str_replace(rownames(klsa_df),"text","")
p1 = klsa_df %>% ggplot(aes(x = V1, y = V2), label= Name) +
  geom_point(size = 2, alpha = 0.6, color = verses.color) +
```

```
  theme_bw()+
  geom_text(aes(label=Name,hjust=1, vjust=1))
p2 = klsa_df %>% ggplot(aes(x = V1, y = V3), label= Name) +
  geom_point(size = 2, alpha = 0.6, color = verses.color) +
  theme_bw()+
  geom_text(aes(label=Name,hjust=1, vjust=1))
cowplot::plot_grid(p1,p2, nrow = 1)
```

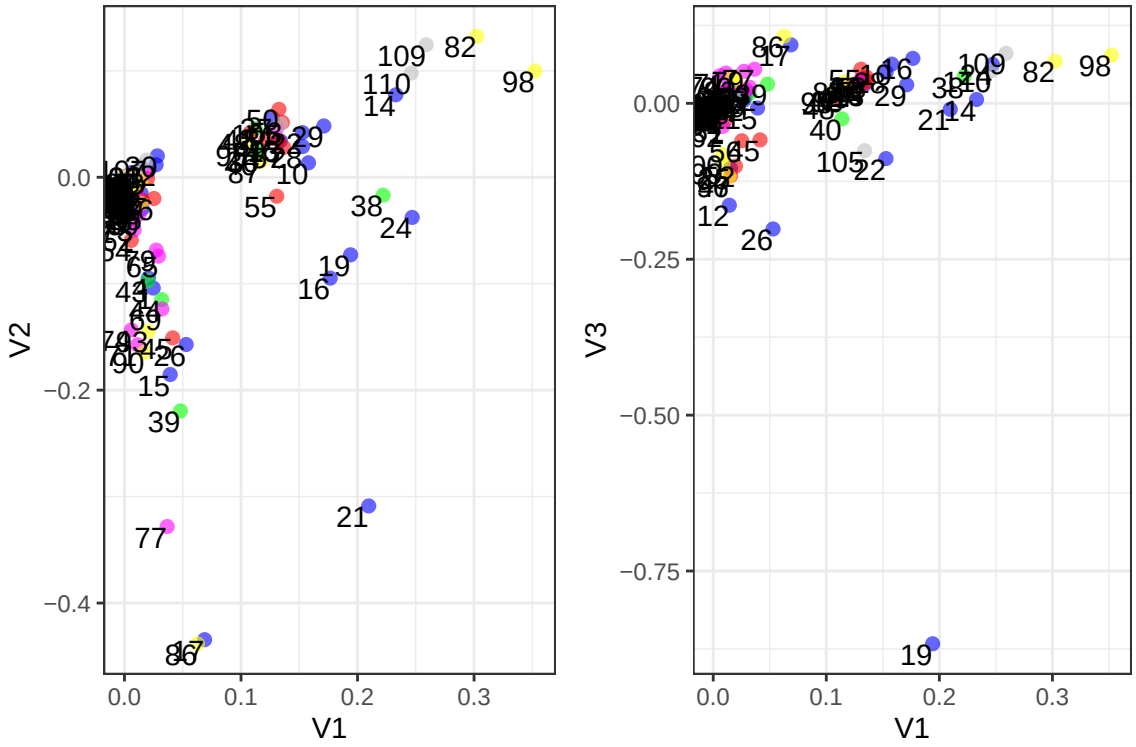


Figure 206: Topics in dimensions for Surah Al-Kahf

In Figure 206, there are (supposed to be) six groupings of dimensions (which are topics). Semantically, we can see that on Topic 1 and Topic 2, as well as Topic 1 and Topic 3, the groupings and the distances between the groupings are not as clear and lumpy in nature. This is due to the fact that we "force" the number of topics to be six by choice. This is the problem of choosing parameters for the model because it dictates the final results based on the assumptions we use.

Now let us present the results from another perspective, that is to view the scores for the topics across the verses. Since it is not easy for us to print the scores and visualize them, we plot the scores in a 3D plotter and present the plot output in Figure 207.

The plot shows that for Topic 1 (the highest-ranked topic), there are verses that have high positive scores. For Topic 5 and Topic 6, almost similar verses have high positive scores, while for Topic 3, the scores are highly opposite (i.e. negative) on some of the verses. This is not exactly the ideal method to extract the information from the model, since visualization of the complex dimensionality is not easy and clear. However, what we want to demonstrate is there are deeper level complexities that are not easy to identify by just eyeballing the visuals.

8.2.8 Summarizing unsupervised learning model

Let us summarize the lessons learned from the LDA, STM, and LSA models of Surah Al-Kahf.

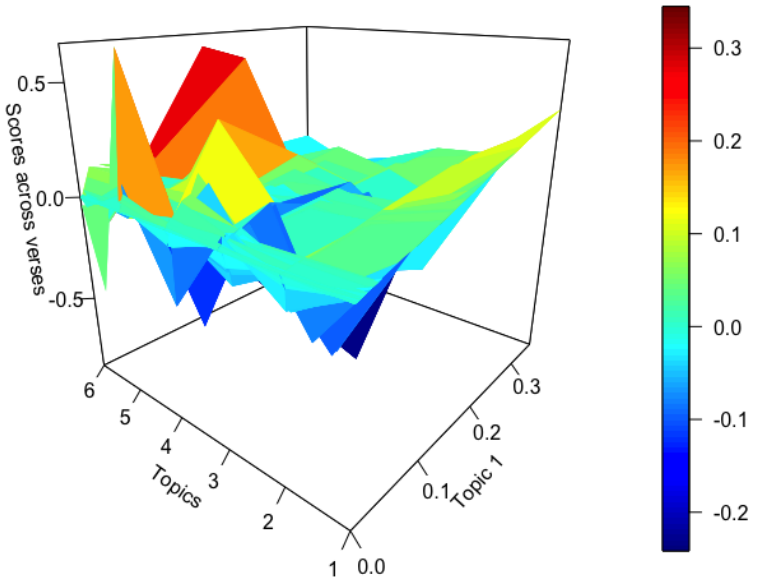


Figure 207: 3D plot of the scores from LSA model for topics in Surah Al-Kahf

Firstly, we note that for the Surah Al-Kahf, we have 507 unique tokens (or vocabulary) from 110 documents (verses), which means that the matrix of data space of 55,770 observations (507 x 110), which is a sparse matrix of 0's and 1's. In the examples we went through, we are trying to estimate 3,549 parameters ((6+1) x 507). Statistically, we are facing an extremely small data set (due to sparsity) while trying to estimate a large number of parameters. This is the first part of the problem we are facing.

Secondly, we need to mention a few notable problems. Is it justified to assume that words, lexically and semantically follow certain patterns and structures within Saheeh English translations of Al-Quran? In the English language, there are abundant works by linguists to deal with this issue. However, if the texts are translations from another language, which has its own grammatical, lexical, and semantical meaning, does the same thing hold? Naturally, the answer is not necessarily. This is particularly true for Quran Analytics since the narratives, stories, and expositions in Al-Quran are explicit and not in the normal human linguistic way.

Summarizing the observations and results of the LDA, STM, and LSA models, we can generally say that the unsupervised learning model represented by the three, despite the sparse data situation, performed reasonably well in capturing some elements in the structure of the texts from statistical perspectives. The model variations could very well result from a small sample problem.

Generally, we would say that, if more analysis is done, like, instead of relying on a single Surah (as we have done here), we could use the entire corpus of Saheeh, and augment the data with other translations (Yusuf Ali and other English translations available). This may help deal with the small sample/data problem. Furthermore, if data augmentation involves the original Arabic text combined with some refinement of the modeling approach, we believe that some interesting insights are possible. This is the more comprehensive Quran Analytics approach that we target. For now, we leave the subject as directions for future research.

8.3 Supervised learning models

A supervised learning model requires two sets of data, the training sample and the validation sample. Both samples must be labeled data, where the categories are pre-labeled, obtained from some other pre-worked dataset, or the user's own labeling. Since we have attempted to label Surah Al-Kahf with six labels (i.e., topics), we want to test some of the supervised learning models and compare the performance and results.

We will choose the simplest method, Naive Bayes (NB), followed by Support Vector Machines (SVM). We will then compare the methods at the end of this section.

First, we create the labels for the data and split the samples for training and validation set as follows:

```
label_index = data.frame("index" = 1:110,
                         "topics" = c(rep("cave",(31-1+1)),
                                      rep("garden",(44-32+1)),
                                      rep("life",(59-45+1)),
                                      rep("moses",(82-60+1)),
                                      rep("dhul",(102-83+1)),
                                      rep("deeds",c(110-103+1))))
train_index = sample(label_index$index,55)
kahf_full = data.frame("text" = kahf$saheeh, "label"= label_index$topics)
kahf_train = kahf_full[train_index,]
kahf_valid = kahf_full[-train_index,]
label_train = kahf_train$label
label_valid = kahf_valid$label
dfm_train = kahf_train$text %>% tokens(remove_punct = T) %>%
                tokens_tolower() %>%
                tokens_remove(pattern = stop_words$word, padding = F) %>%
                dfm()
dfm_valid = kahf_valid$text %>%  tokens(remove_punct = T) %>%
                tokens_tolower() %>%
                tokens_remove(pattern = stop_words$word, padding = F) %>%
                dfm()
```

8.3.1 Naive Bayes (NB)[110]

The Naive Bayes model for text classification is a simple "Bag-of-Words" model, where we assume a (prior) multinomial probability for six topics, which we assume by default. Each verse is assumed to have an equal probability of belonging to a topic. The model then will calculate the probability of a verse belonging to any one of the topics. By default, we know which verses are from which topics, as defined in the six labels that we have set before.

The Naive Bayes model in *quanteda* is simply fitted and tested as follows:

```
nb_kahf = quanteda.textmodels::textmodel_nb(dfm_train,label_train)
table1 = table(predict(nb_kahf),label_train)
table1
dfm_matched <- dfm_match(dfm_valid, features = featnames(dfm_train))
pred = predict(nb_kahf, newdata = dfm_matched)
table2 = table(pred, label_valid)
table2
```

We can see from the results in the confusion matrix table, *table1*, that in the training set, the Naive Bayes model has a perfect fit (almost all correct matches) with the accuracy of 100%. However, when fitted with the balance of the data (validation data), there are several mismatches, and the accuracy drops down to 50.91%. We have an overfitting problem.

[110]For reference, please refer to https://quanteda.io/reference/textmodel_nb.html

8.3.2 Support Vector Machines (SVM)

Support Vector Machines or SVM is among the basic multivariate regression models which are based on separating hyperplanes concept. The idea of SVM is to generate classifications across multi-labels by separating the data into separate "planes" of classes.

In *quanteda*, the model can be applied using *textmodel_svm()* function.

```
svm_kahf = quanteda.textmodels::textmodel_svm(dfm_train,label_train, weight = "uniform")
table3 = table(predict(svm_kahf),label_train)
table3
pred = predict(svm_kahf, newdata = dfm_matched)
table4 = table(pred, label_valid)
table4
```

We can see from the results of the table that in the training set, the SVM model fits with an accuracy of 100%. However, when fitted with the balance of the data (validation data), there are several mismatches, and the accuracy is at 4545%. It suffers the same problem as the Naive Bayes model, which is an overfitting problem.

8.3.3 Summarizing supervised learning model

Our excursion on the supervised learning model is very short, with unsatisfactory results. The problem we face is quite obvious, namely small sample data. We have only a few verses for each topic, and within each topic, we have only a small number of features. When we fit the model, it will easily get overfitted, which to a degree says that our labeling of the verses is precise within its own dataset whereby the model would detect it easily. However, we can never use the learned model to apply to verses beyond the actual training sample, which then renders the exercise practically useless.

8.4 Ideological difference models

quanteda.texmodels has a few other supervised learning models, such as *textmodel_wordscores()* and *textmodel_wordfish()*. Both of these models are useful for observing the "ideological" differences in texts. The concept of both models is to uncover if there are any differences ideologically speaking between the various topics. If there are "distinct" ideas, the model should be able to predict them. These types of models are in the category of "ideological scaling", a supervised learning model (please see Figure 197) at the beginning of the chapter.

First, we will apply the *wordscores* model in the codes attached below, and tabulate the results.

```
ws_kahf = textmodel_wordscores(dfm_train,
                               y = as.numeric(as.factor(label_train)))
table5 = table(floor(predict(ws_kahf)),
               as.numeric(as.factor(label_train)))
table5
table6 = table(floor(predict(ws_kahf, newdata = dfm_matched)),
               as.numeric(as.factor(label_train)))
table6
```

From the tables printed above, we can say that in the training part *(table5)*, the model could clearly identify that the "ideas" within each topic are distinct enough to be detected with the accuracy of 38.18%. In *table6*, when we apply the model to the validation data, we can see that some of the topics (ideas) are dispersed through other topics (ideas).

```
textplot_scale1d(ws_kahf, margin = "features",
                 highlighted = c("allah","lord","moses","cave",
                                 "al-khidh","gardens"),
                 highlighted_color = "darkred")
```

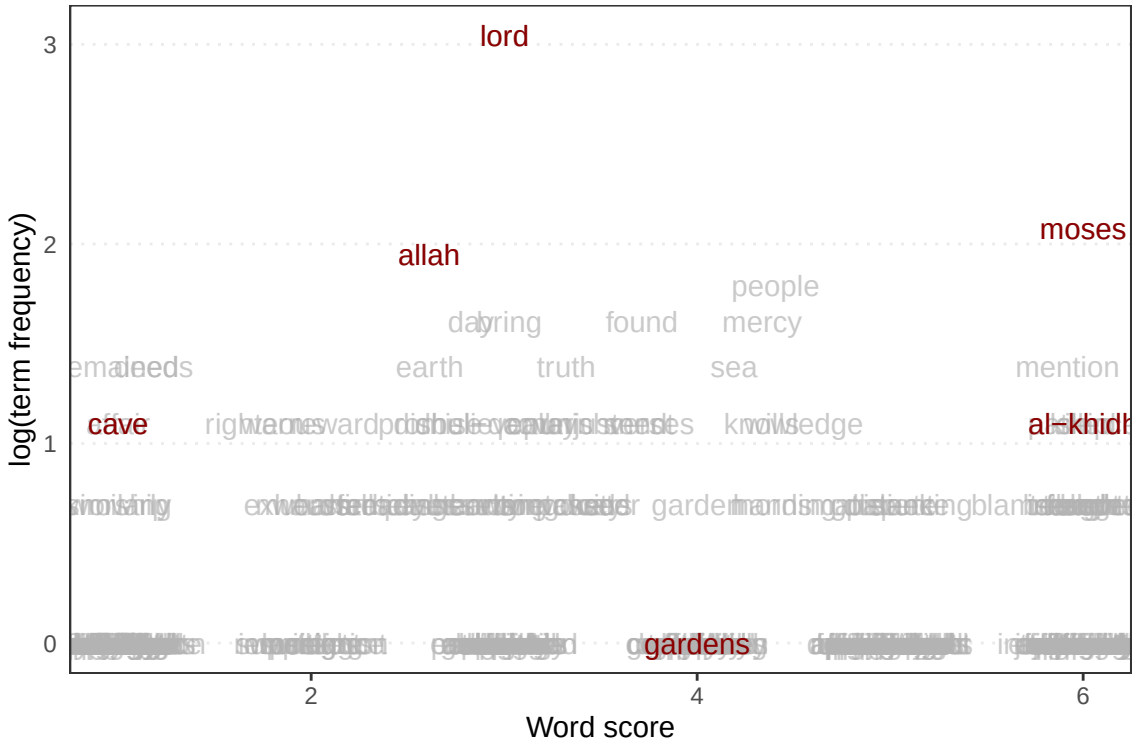


Figure 208: Wordscores plot for Surah Al-Kahf

Now we will plot the scores from the model together with some words to be highlighted. The plot is shown in Figure 208. From the plot we can visualize the relative positions of the key words for the various topics: "lord", which rank highest in the Surah, followed by "allah" - both almost at the center; the word "moses" scores higher, but in the same "direction" as "al-khidh", which is directly below "moses". The word "cave" and "gardens" are figuratively apart. Similar exercises can be performed for various selections of key terms and we can observe its relative position within the whole text.

Now we will apply *textmodel_wordfish()*, which in many ways, similar to *wordscores* model and plot the results.

```
wf_kahf = textmodel_wordfish(dfm_train, dir = c(1,2))
textplot_scale1d(wf_kahf, margin = "features",
                 highlighted = c("allah","lord","moses","cave",
                                 "al-khidh","gardens"),
                 highlighted_color = "darkred")
```

wordfish scores which will show whether some of the topics are "diametrically" opposed to one another. Figure 209 demonstrates that all topics are pretty much "aligned" to each other from the word "lord", down to "gardens". An interesting observation is that "moses" is right after "lord" in the ranking, above "allah".

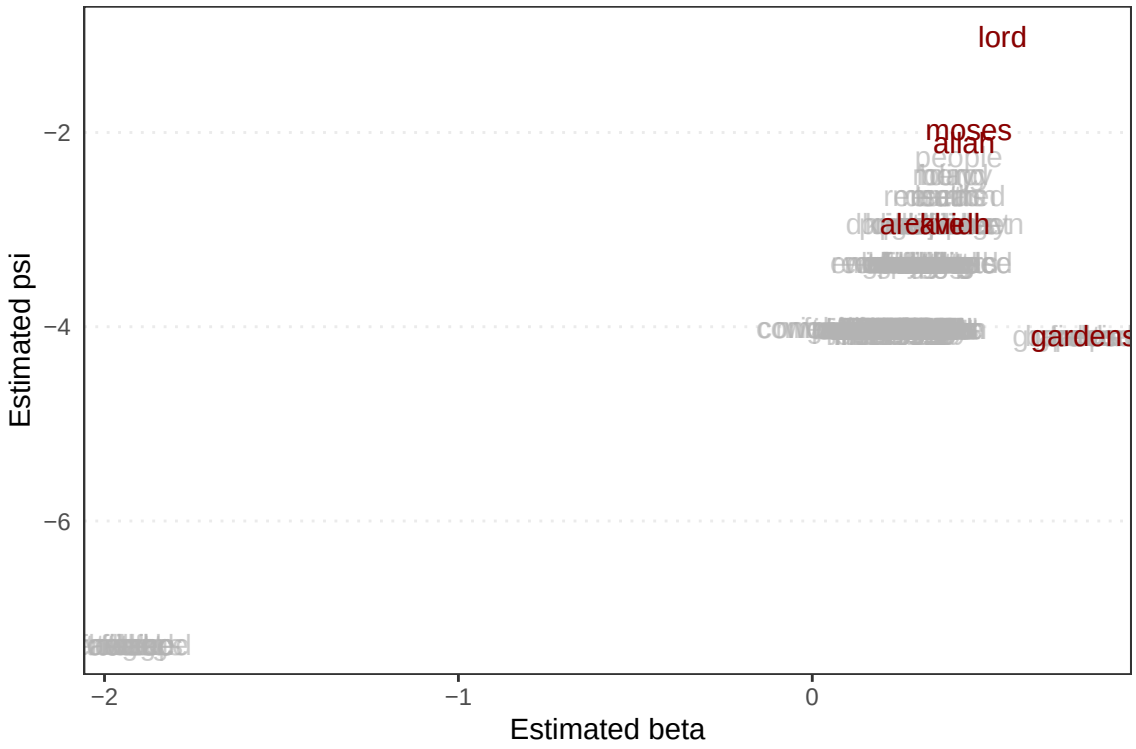


Figure 209: Wordfish plot for Surah Al-Kahf

8.5 Word embeddings models

In this section we will introduce a powerful approach to dealing with text data, using a text embedding method known famously as GloVe: Global Vectors for Word Representation (Pennington et al., 2014). It is "an unsupervised learning algorithm for obtaining vector representations for words". The concept of word-vector representations is to develop a global word-word co-occurrence matrix that tabulates how frequently words co-occur with another one in a given corpus.[111] The model first sets a pre-training on all the data within the corpus, allowing any utilization thereafter to be easy and fast.

Mathematically what the GloVe algorithm does is to transform the corpus into a "flat" and "compact" matrix of features (row-wise) vectors. Each feature is represented as a vector of fixed length (known as the dimension) set by the algorithm. The concept is similar to hashing algorithms, where each vector is unique (representing a unique feature or word in the vocabulary). Normally the length of the vector is set to fifty, which is deemed to be sufficient for most large-size tasks. We introduce the concept here for purposes of illustrating the usage and convenience of the algorithm and demonstrate its potential for Quran Analytics.

In **R**, the GloVe algorithm is implemented through the *text2vec* (Selivanov et al., 2020) package.

The steps in *text2vec* are as follows:[112]

1. *space_tokenizer()* and *itoken()* for tokenization of the corpus or texts, which includes any removal of unwanted items (stopwords, punctuations, etc.)
2. *create_vocabulary()* to create the vocabulary for the entire tokens (i.e. unique tokens)
3. *vocab_vectorizer()* to vectorized the vocabulary
4. *create_tcm()* to create the term-co-occurrence-matrix (tcm) and set the skip grams window

[111] https://nlp.stanford.edu/projects/glove/

[112] The model is compiled in C++ language and wrapped into **R**; which provides fast speed of computation and memory-efficient processes.

5. *GlobalVector$new()* to generate the matrix of vectors for each vocabulary item, and set the length of the vector (to 50)
6. *xxx$fit_transform()* to fit the GloVe model
7. Use the model to predict

Note that the package uses *data.table* syntax for *data.frame*, which is an extremely fast data processor in **R**.

```
## load the library
library(text2vec)
## clean the texts
quran_saheeh = tolower(quran_all$saheeh)
quran_saheeh = gsub("[^[:alnum:]\\-\\.\\s]", " ", quran_saheeh)
quran_saheeh = gsub("\\.", "", quran_saheeh)
quran_saheeh = trimws(quran_saheeh)
## 1. tokenize
ktoks = space_tokenizer(quran_saheeh)
itktoks = itoken(ktoks, n_chunks = 10L)
stopw = stop_words$word
## 2. create vocabulary
kvocab = create_vocabulary(itktoks, stopwords = stopw)
## 3. vectorize the vocabulary
k2vec = vocab_vectorizer(kvocab)
## 4. create the tcm
ktcm = create_tcm(itktoks, k2vec, skip_grams_window = 5L)
## 5. generate the Global Vector with rank = 50L
kglove = GlobalVectors$new(rank = 50, x_max = 6)
## 6. Fit the GloVE model
wv_main = kglove$fit_transform(ktcm, n_iter = 20)
## Use the model for prediction
wv_context = kglove$components
## This is in data.table format
word_vec = wv_main + t(wv_context)
## This is to convert to dplyr data.frame format
word_vec_df = as.data.frame(t(word_vec))
```

The word-vector consists of vectors of unique frequencies for each word in the vocabulary. These frequencies are used for calculating the similarities or distances between the words. A plot of selected word frequencies in the word-vector is presented in Figure 210, for the word "allah", "lord", "muhammad", and "abraham".

```
word_vec_df %>% ggplot() + geom_line(aes(x=1:50, y=allah), color = "red") +
                    geom_line(aes(x=1:50, y=lord), color ="steelblue") +
                    geom_line(aes(x=1:50, y=muhammad), color = "magenta") +
                    geom_line(aes(x=1:50, y=abraham), color = "green") +
                    labs(x = "dimensions", y = "word frequencies")
```

The frequencies do not have any meaning, except that it records the relative unique position of each word within a corpus. This method is a faster way of generating an unsupervised learning model for the data at hand, especially when the data (i.e., text corpus) is large and sparse.[113]

The codes below show some examples of how the GloVe model is used.

```
topic1 = word_vec["allah",,drop = FALSE] +
         word_vec["lord",,drop = FALSE] +
         word_vec["muhammad",,drop = FALSE] +
         word_vec["abraham",,drop = FALSE]
topic1_sim = sim2(x = word_vec, y = topic1, method = "cosine")
head(sort(topic1_sim[,1], decreasing = T),7)
```

[113]Note that the matrix is a much more compact space than the DFM or FCM matrices we looked at earlier in *tidytext* and *quanteda*.

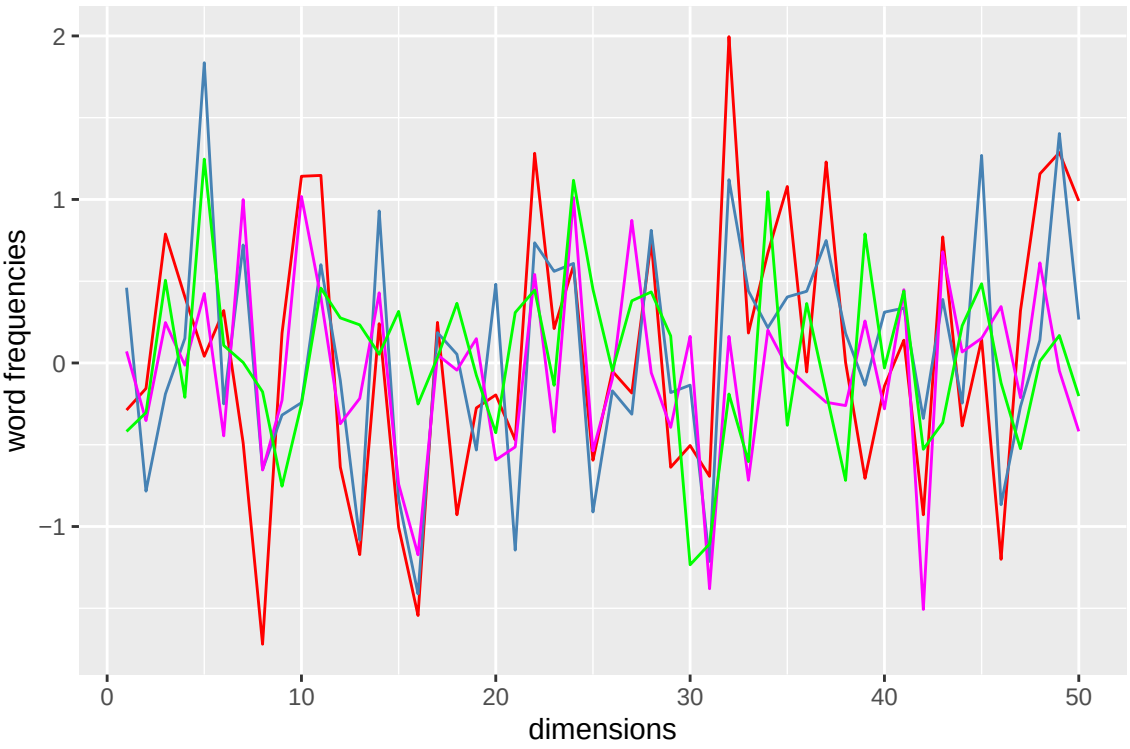


Figure 210: Word-vector frequencies for selected words

Now we can see clearly the topical relevance of the words by rank, from “lord”, to “allah”, then “muhammad”, as a “messenger”, bringing the “truth” to be “believed” by the “people”, and so on. The word “abraham” (Prophet Ibrahim a.s.) appears much further down in the ranking.

Now instead of us “fixing” the topics as in the LDA, STM, and LSA to six topics, we can use the words we think are relevant for the topic and observe the results. Let us imagine the story of the cave dwellers in Surah Al-Kahf, and try to pick out “cave” and “dwell” from the entire translation.

```
topic = word_vec["cave",,drop = FALSE] + word_vec["dwell",,drop = FALSE]
topic_sim = sim2(x = word_vec, y = topic, method = "cosine")
head(sort(topic_sim[,1], decreasing = T),7)
```

As an exercise, let us look at “worship”,“allah” and “idols”.

```
tt = word_vec["worship",,drop = FALSE] + + word_vec["allah",,drop = FALSE]
tt_sim = sim2(x = word_vec, y = tt, method = "cosine")
head(sort(tt_sim[,1], decreasing = T),7)
tt = word_vec["worship",,drop = FALSE] + + word_vec["idols",,drop = FALSE]
tt_sim = sim2(x = word_vec, y = tt, method = "cosine")
head(sort(tt_sim[,1], decreasing = T),7)
```

Let us now go back to Surah Al-Kahf and redo the topic modeling exercise but instead, we will use the GloVe formulations and set the topic to be six (the same as before).

```
kahf_saheeh = quran_all %>%
                filter(surah_title_en=="Al-Kahf") %>%
                pull(saheeh)
kahf_saheeh = tolower(kahf_saheeh)
kahf_saheeh = gsub("[^[:alnum:]\\-\\.\\s]", " ", kahf_saheeh)
kahf_saheeh = gsub("\\.", "", kahf_saheeh)
```

```
kahf_saheeh = trimws(kahf_saheeh)
tokens = word_tokenizer(kahf_saheeh)
it = itoken(tokens, ids = 1:length(kahf_saheeh),
            progressbar = FALSE)
stopw = stop_words$word
v = create_vocabulary(it,stopwords = stopw)
v = prune_vocabulary(v,
                     term_count_min = 3,
                     doc_proportion_max = 0.2)
vectorizer = vocab_vectorizer(v)
dtm = create_dtm(it, vectorizer, type = "dgTMatrix")
set.seed(12345)
lda_model = text2vec::LDA$new(n_topics = 6,
                              doc_topic_prior = 0.1,
                              topic_word_prior = 0.01)
doc_topic_distr =
  lda_model$fit_transform(x = dtm, n_iter = 1000,
                          convergence_tol = 0.001,
                          n_check_convergence = 25,
                          progressbar = FALSE)
```

Figure 211 shows the prominence of the topics in Surah Al-Kahf. We can see better which topics rank higher, in fact, only a few of the topics are prominent compared to others.

```
barplot(doc_topic_distr[1, ], xlab = "topic",
        ylab = "proportion", ylim = c(0, 1),
        names.arg = 1:ncol(doc_topic_distr))
```

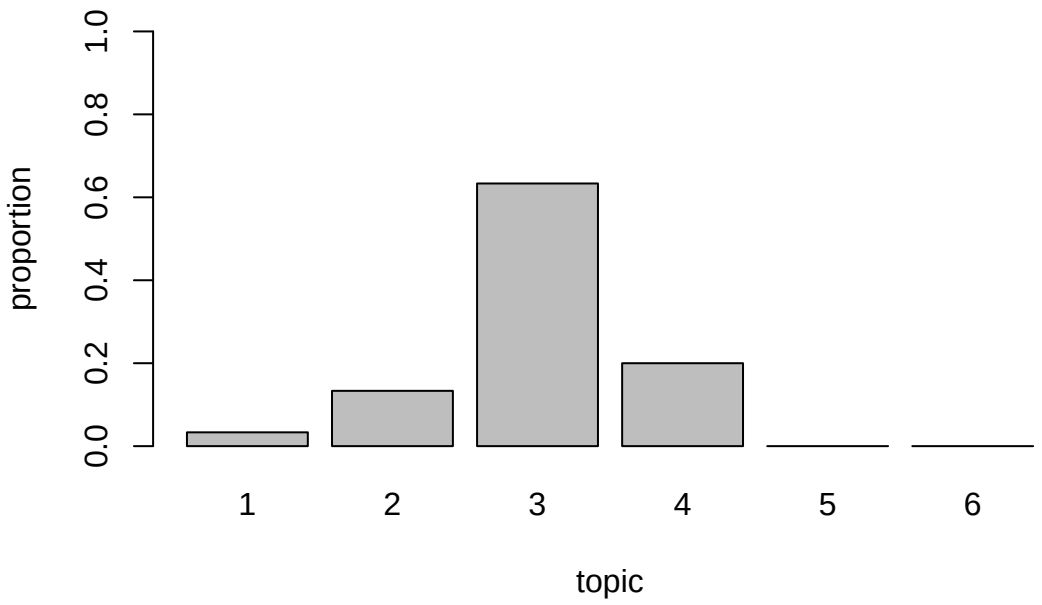


Figure 211: Topic bar plot for Surah Al-Kahf using GloVe model and LDA

We can get the top words for each topic, sorted by probability ranking as follows:

```
lda_model$get_top_words(n=7, topic_number = c(1L,2L,3L,4L,5L,6L), lambda = 1)
```

Without knowing what are the topics in Surah Al-Kahf, it is amazing to see that indeed the six topics can be seen from the words: cave dwellers from Topic 5, the two owners of the gardens in Topic 3, Moses and Al-Khidh in Topic 6, and Dhul Qarnayn in Topic 4.

We will fit the LSA model one more time, using the GloVe algorithm, and plot the results.

```
lsa_model = text2vec::LSA$new(n_topics = 6)
doc_topic_distr =
  lsa_model$fit_transform(x = dtm, n_iter = 1000,
                          convergence_tol = 0.001)
```

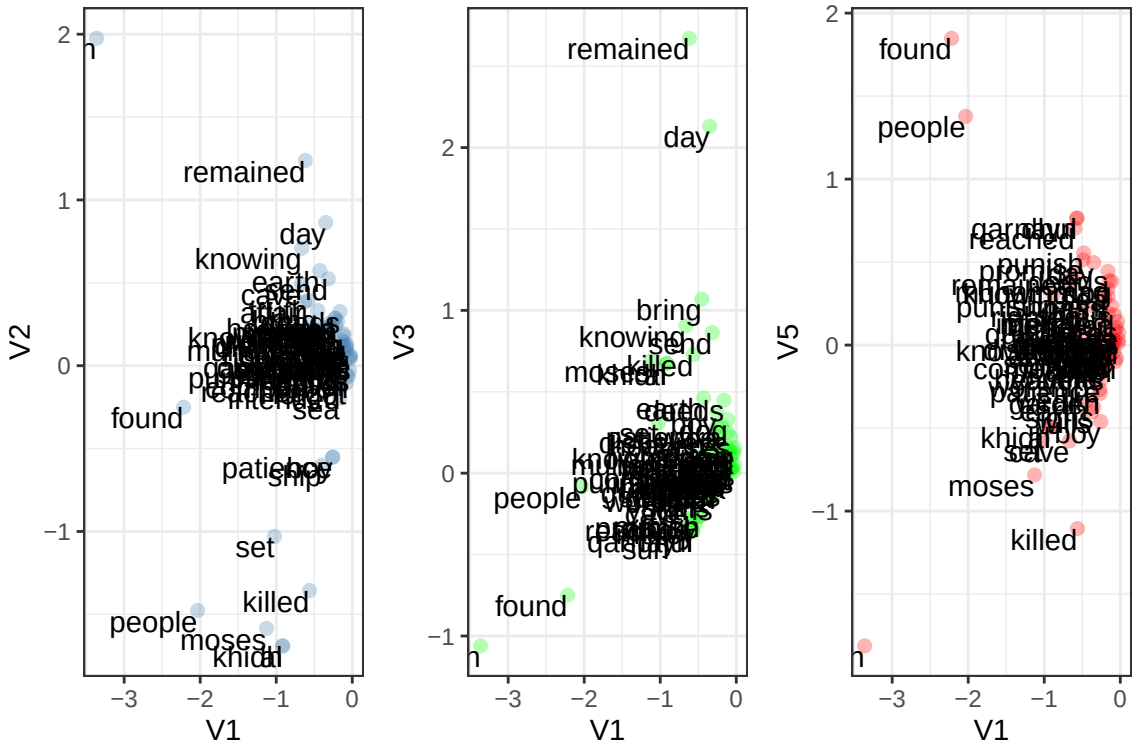


Figure 212: Topics in dimensions for Surah Al-Kahf using LSA and GloVe

The results in Figure 212 are different from the ones in Figure 206, where instead of the verses, we plot it over the words. As noted in many experiments using the LSA model, while it can generate distinctions between the topical relations, it is very hard to interpret the output. We can see that more "verbs" (such as "remained", "found", "killed") appear to be further from the main clustering, which semantically carries more meaning than just proper nouns or names.

8.5.1 Summarizing word embedding model methods

The examples from this section on word embeddings using GloVe algorithms show promising results and demonstrate the strength of the method. It is a new generation of unsupervised learning methods for NLP tasks of finding topics, analyzing various language structures, and many others. The strength of GloVe lies in its simplicity and reliance on "closed and compact" space representations of text data, which allows us to deal with the problems of the Power Law distribution of Zipf's law and many other statistical anomalies in text analysis. The strength of GloVe is proven by the fact that it is used heavily by Google (as it was originally developed together between Google and Stanford NLP Group).

8.6 Summary

The chapter explored the subject of text modeling by traversing through unsupervised and supervised learning models. We have demonstrated the uses and benefits of the models by applying them on topic modeling tasks applied on Surah Al-Kahf. Generally, we would say that any modeling for the English translations of Al-Quran using these models will suffer from "small sample" problems, due to sparsity structures within the data.

In the case of the supervised model, despite the shortcoming of sample size problems (i.e., model overfitting), there is still a high potential of usage if annotations and labeling are obtained correctly. These annotations are available in the form of exegesis of Al-Quran, such as the Tafseer of Ibnu Katheer and others. The annotations can be used as training labels and hence the model can be improvised to better predict the data. This is one area of future research for Quran Analytics.

On the other hand, classical and older versions of the unsupervised learning model, such as Latent Dirichlet Allocation (LDA) and Latent Semantic Analysis (LSA) are a bit arcane in the results; despite having potentials of further development along the lines forwarded by the model, conceptually. For Structural Topic Models (STM), there are many other potentials for improvements, since it has many flexibilities for changing and altering modeling assumptions.

We have not covered another large area of supervised and unsupervised learning using Neural Network models, as well as Deep Learning models of Neural Networks. Similarly, there are also powerful models based on Hidden Markov Models (HMM), which we have not covered. HMM has been extremely successful in speech recognition modeling. All these areas are very large and require extensive work for which we envisage our Quran Analytics project to undertake.

Lastly, we introduced the Word-to-Vector model of GloVe algorithms, which is an extremely useful tool with wide application potential. This is an example of a new generation of unsupervised learning models which is gaining popularity. Combining GloVe with Deep Learning is of course a venture that is among the latest in NLP research.

Since there are too many areas to cover, we end the chapter by concluding that the area of applications of text models, machine learning models (supervised and unsupervised) in Quran Analytics is just at the beginning, there are lots more to be discovered.

8.7 Further readings

quanteda package documentation (https://quanteda.io).

LDA package references from (Blei et al., 2003).

LSA package references from (Landauer et al., 1998).

STM package references from (Roberts et al., 2014) and (Roberts et al., 2019).

text2vec package references (http://text2vec.org/index.html)

GloVE references (https://cran.r-project.org/web/packages/text2vec/vignettes/glove.html)

9 Knowledge Through Verse Network

Knowledge graphs (KG) have been a major area of research since 2012, after its introduction by Google when they enhanced their search engine as a semantic search, upgrading from strings search process (Singhal, 2012). Formally defining knowledge graphs, however, is challenging, since almost any form of representations of relationships, mathematically can be structured fitting a graph definition. Generically, it is "a network of all kinds of things relevant to a specific domain, not limited to abstract concepts and relations but can also contain instances of things like documents and datasets"(Blumauer, 2016).

The early development of knowledge graphs was focused on gathering and organizing information from the web onto a knowledge graph framework. The leaders were Google, Yahoo, and major search engines of the web. Later on, the development moved towards knowledge extraction processes from semi-structured web knowledge bases such as Wikipedia, and towards the development of Semantic Web as well as the development of schema.org (Paulheim, 2016). The main challenges involve the methodologies of building the "graph from knowledge", and "defining knowledge itself".

Within the field of Information Retrieval(IR), the application of knowledge graph is growing, due to the instrumental role of semantic search enabling of KG. Within IR, among the key tasks are understanding queries and documents, matching and returning direct and clear answers as well as actionable entities or relationships, for any form of subject or domain of interest. The main issue facing both subjects is the intertwining directions between "knowledge graphs for information retrieval" and "information retrieval for knowledge graphs"(Reinanda et al., 2020). In graph theory, this problem is couched in the term of reference for a graph, defining which are the nodes and which are the edges. For example, do we represent "documents" as nodes and "entities" as edges or the reverse.[114]

The subject of KG, IR, and combinations of both is among the latest research works which provide advancements in knowledge theory, in particular towards building expert systems and knowledge retrieval. Quran Analytics as we envisage is aiming towards creating an open system for Quranic knowledge retrieval and related learning systems as the end goal.

The first and foremost challenge in creating KG and developing IR systems on top of it is "annotating" the knowledge. Interestingly though, within Islamic sciences, "annotating" the knowledge has been practiced by Islamic scholars from its very early time as a very strict discipline. We can see how the classical books are documented in an extremely meticulous way linking information, in forms of interpretations of verses of Al-Quran and the sayings or traditions of the Prophet (s.a.w). This is evident in the compendium of exegeses, such as by Imam Ibnu Katheer (Tafseer Ibnu Katheer), Imam Al-Razi (Al-Tafseer Al-Kabeer, The Large Commentary), and Imam Al-Tabari (Tafseer Al-Tabari). In Al-Hadith, we have the Sunan Sittah (the six major compendiums of Al-Hadith), which disciplines were established by Imam Malik (in Al-Muwatta'), and Imam Al-Bukhari (Saheeh Al-Bukhari) followed by Imam Muslim (Saheeh Muslim), and others.

The knowledge of these classical Islamic scholars (and many others for that matter) has been "preserved" in their annotations, labeling, documenting the source, interpretation, and links as provided by the writings and records of the Classical texts of Islamic sciences. What they had established in essence is knowledge retrieval systems, combining both the methodologies of KG and IR, and resolving some of the major issues between the "documents" and "entities" - where the entities are the source of knowledge (for example the narrators in Al-Hadith), and the documents (for example the Matan, or body of the Hadith).

The task for Quran Analytics, therefore, in the first instance is to convert all these "information" into a graph framework, as envisaged by the Knowledge Graph concept, to allow us to "extract knowledge" from the sources (such as from the verses) and transform them into learning. Since the field is extremely vast and challenging, the first focus of our Quran Analytics project is on the major exegesis of Al-Quran.

Another interesting issue is, how do we learn what are in the "mind" of the classical scholars of exegesis of Al-Quran? Many of their ideas and thoughts are actually "hidden" from a direct naked reading of their writings. It would be almost impossible to memorize and link all concepts and aspects presented by these scholars, especially ideas which are not "explicitly written", except through annotations. For example, when a reference is made, say for a verse that is interpreted by other verses, the reasoning for such references may be inexplicitly stated, for reasons known only to the author. Uncovering this "hidden knowledge" is exactly what later scholars did when they relied on the classical sources and combined them with their own knowledge. (i.e., interpreting interpretations with their own interpretations). We can see this in the work of Imam Ibn Hajar, for example, in providing a detailed exegesis of the Sahih Al-Bukhari. This is exactly what is happening to the translations of Al-Quran, which in essence is actually interpretations of Al-Quran, into another language by the author.

Among the major requirements for Quran Analytics is to build "translation engines" for "interpretations"

[114]Inverting a graph is not a simple and direct mathematical process.

(i.e. translations) of Al-Quran in other languages, including Modern Standard Arabic language,[115] besides the obvious commonly used languages such as the English language and other major languages. In the case of the Malay language, the authors' mother tongue, the need is rather acute, at least based on our own experience of learning Al-Quran. The distance between Malay speaking people with their own mother tongue is diverging, the distance with the Arabic language is even wider, and the teaching of Al-Quran is relegated towards memorization and less towards understanding.[116]

9.1 Tafseer Ibnu Katheer as Knowledge Graphs

Tafseer Ibnu Katheer is a classical Quranic exegesis, using a methodology that is considered by many scholars as to the "gold standard" of interpretations of Al-Quran, that is "Interpretation of the Quran with the Quran" (Tafseer Al-Quran bi Al-Quran).[117] What Ibnu Katheer did was to provide links between a verse to other verses which interpret the verse. By doing so, what is documented is his views on how each verse is linked to other verses, and in turn, implicitly, these verses are also linked to other verses and so on. The network of links between these verses then forms a massive network of links between all possible verses to all other possible verses.

In network terms, we say that each verse is a "node", and each of the links is an "edge", and the total network formed is a "graph". The information provided by the graph is called a "knowledge graph". In this case, the full knowledge graph of the Tafseer represents what Imam Ibnu Katheer recorded as his way of interpreting not only just each verse of Al-Quran but the entire verses of Al-Quran in totality. Whether that had been implicitly intended by him, or it is what his complete understanding of the whole Al-Quran is not explicitly expressed. Once we convert his whole interpretation into the form of a network as described, we can see the structures of his "knowledge" and get the deeper meanings of the knowledge representations of his methodologies and hence its deeper and broader understanding.

In this chapter, we will do an expanded analysis of Tafseer Ibnu Katheer, using the tools and methods explored in previous chapters.

Open source network visualization tools are handy when we deal with large networks. There are several open-source tools available such as *Gephi*[118], *Cytoscape*[119], and *Pajek*[120]. For interested readers, please refer to the manuals in *Gephi* for the methods of applying the visualizations presented here.

The codes used for this chapter are in the Appendix.

9.1.1 Preparing the data and settings

We will start with general network visualizations. Network visualizations are useful for general analysis where the objective is to observe any "emergent" structures discernable from the observed patterns from the network. Here we will utilize plotting packages from **R**, namely *ggplot2*, and *ggraph*; we will also deploy *Gephi*, which is open-source software for large-scale network visualizations and network statistical computations.

Furthermore, we will use some of the tools from previous chapters to do deeper analysis on the network data, and explore some examples of "deep diving" into a Surah, or a verse, or a group of verses. We will use the graph theory and visualizations in Chapter 5, Chapter 6, text modeling techniques of Chapter 7, and Chapter 8. In essence, we will glue together the works from previous chapters into our analysis.

[115] Our views as to why the need applies to MSA is due to the fallacious assumptions that Arabic speaking people knows the Quranic language. The subject is controversial as far as we know since it is a subject imbued within the domain of linguistic and Islamic knowledge.
[116] The authors' personal views.
[117] Tafseer Ibn Katheer, https://en.wikipedia.org/wiki/Ibn_Kathir
[118] https://gephi.org
[119] https://cytoscape.org
[120] http://mrvar.fdv.uni-lj.si/pajek/

The dataset which provides the annotations for the Tafseer Ibnu Katheer is from http://textminingthequran.com/.[121] The dataset contains 7,679 rows and two columns, a column for the reference verse and the "interpreting" verse in the second column. These two columns are the "vertices" or "nodes" and a row represents an "edge".

Notationally speaking, we have $K_G = set(N, E)$, where K_g is the *Katheer Graph*, and N are the nodes (i.e., verses) in the graph, and E are the edges or directed links (i.e., the relationship) between the nodes (verses), which therefore means that the graph is a *directed* graph.[122]

Using *igraph*, we convert the dataset to a graph object and we are ready to explore the Katheer Graph.

9.2 Katheer Graph network

9.2.1 Katheer Graph visualizations

In network analysis, we use various methods of display or visualization of the network, which is called the layout. What each layout does is expand the network visually for visual observations. Here we provide a few samples of visuals and followed by short interpretations.

1. *Fruchterman-Reingold* layout visualization

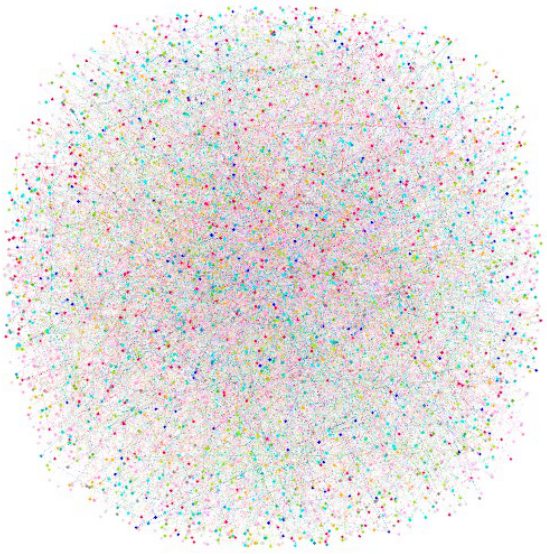


Figure 213: Ibnu Katheer verse network: Fructherman Reingold layout

First, we calculate the modularity classes for the network (based on a resolution scale of 1.0) and color each class distinctly. As observed in the image, the various 115 communities are spread out throughout the verses in the entire Al-Quran. What this implies is that the method of interpretations of Ibnu Katheer is expanding the grouping of verses which are interpreted by each other (i.e. networked with each other), spanning across a large space encompassing many verses from other parts of Al-Quran (e.g., from other verses within the same Surah as well as from other Surahs). The visuals also imply that it took many verses to explain a verse, or in another way, many verses linked together give a much wider interpretation of the verse, and vice-versa.

This is indeed a magnificent view of what Ibnu Katheer had in mind. If we want to this manually, it will take a great effort to annotate and make notes of each verse, grouping, and link altogether.

[121] Note that we have not verified the accuracy of the dataset, and we will use it as is where is basis.
[122] This is a formal definition of a graph in Graph Theory

The largest modularity class, which stood at 5.65% of verses (nodes) is a group of verses (community) consisting of about 300 verses. What it normally means is that these verses are from a common theme or subject. If we want to know what the themes are, we must extract the entire "sub-network" and do the types of analysis which were shown in earlier chapters (word network, co-occurrence network, topic models, sentiment analysis, etc.). Furthermore, the sub-network itself may consist of smaller sub-networks. We can keep repeating the process until we reach its most elementary form. Since our work here is for exploratory purposes, we will leave these details for future work.

2. Add *Force Atlas 2* layout to existing graph

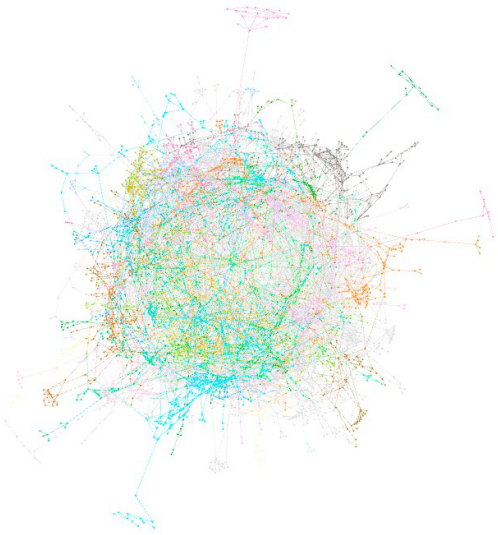


Figure 214: Ibnu Katheer verse network: Force Atlas layout

Now we can observe that if we "re-arrange" the visualizations (by adding the Force Atlas algorithm), the groupings are more distinctly clearer. What this means is those different methods of visualizing provide views of various dimensions of the network. In fact, there are numerous methods of layout layerings that are easily applied within a network structure. All these layouts are based on certain "graph spring" algorithms, whereby each algorithm (logics) brings in another dimension to the graph (or knowledge representations). Again, we leave the subject for future work.

9.2.2 Katheer Graph network statistics

First, we summarize the network properties of *Katheer Graph* and comment on its general structure.[123]

Katheer Graph Network	Statistics
Total number of possible nodes	6,236
Number of source nodes	2,445
Number of target nodes	3,279
Number of edges	7,679
Average degree	2.46
Average path length	14.82
Diameter	51
Modularity (resolution 1.0)	115

[123]References for a mathematical and clear understanding of network structure and its properties is available in Barabasi (2016).

Katheer Graph Network	Statistics
Node average clustering	0.064

A quick observation from the table tells us that not all verses from the 6,236 verses are represented, or we can say that there are nodes without any edges. Mathematically speaking we have 19,440,730 possible one-to-one combinations between the verses. We have only 0% of the possible links, which means that the network is "linearly sparse" in the first layer.

This implies that Ibnu Katheer viewed that not all verses are linked directly to other verses, and possibly a link or few links are sufficient to transmit the message. However, the average path length tells us something else, the message is carried through quite a long path, which on average is 14.82 "hops" or "steps". In another word, a path, denoted by p is represented by: $V_{p,n} > V_{p,n-1}$ ...> $V_{p,0}$, representing a long message passing process. Verse $V_{p,n}$ is interpreting Verse $V_{p,0}$ through a sequence of verses. Note that the number of possible paths can be exploding exponentially (i.e., $i = 1, 2, ...m$).

Are these path sequences random or are they part of the inherent structure of Ibnu Katheer? If the sequence is random, then message passing is also random, which logically cannot be true. If it is true, then we can say that Ibnu Katheer's method does not have "knowledge value". We will show later that true enough to the status of its exegesis, the brilliance of Ibnu Katheer is par excellence beyond imagination. Instead of having to annotate the links for each of $V_{p,n}$, $V_{p,n-1}$,.. to $V_{p,0}$ directly, the message in $V_{p,n}$ can be traced through the paths to $V_{p,0}$. However, a beginner learner needs to read the Tafseer and traverse through all those available paths, many times before he can learn the full message contained in a verse.[124] We know this by the fact that the largest p is 51, the *diameter* of the network.

Given such an enormous structure of the Katheer Graph or knowledge network of verses, it deserves a thorough analysis, which we will do subject by subject next.

9.2.3 Katheer Graph network degree

The degree of the network represents the concept of "links between ideas" (we will use "message", instead of "idea"). If a verse is a "link" to another verse, it means that the "message" contained in the "source" verses explains the message in the "target" verse. The network shows that on average, a verse is connected to (interpreted directly by) 2.46 verses. However, as normally understood in statistics, an average is meaningless without understanding the shape of the entire distribution; which is shown in Figure 215.

The scatter plot of the distribution (Figure A) and the shape of the (log-log) ranked degree distribution of the network (Figure B) proves one key point: the network degree distributions seem not to conform to Zipf's Law. This is in contrast to normal word and word co-occurrence networks (see discussions in Chapter 2). If the network does not follow a Power Law distribution structure, then what is the form of the network structure?

To make sense of this phenomenon, just think of the citation network for scientific literature, where each verse is a journal and the links are the citations.[125] What we observe for citation networks is the case where only a few journal papers are cited with high frequencies and many papers with few or little citations or no citations. Those papers with a high number of citations hold some prestigious information (or knowledge), whilst the others are of lesser importance. The shape of a typical citation network degree distribution is a downward curved "L" shape. But what we see here for *Katheer Graph* is an extremely amplified version of a citation network; where many verses are cited by at least one other verse, and yet, if we trace further, those verses which are not cited directly are cited through the next layers of the network (as explained in the "paths" discussion before).

[124] In database search terms, we call this sequential search process. Sequentially, this process will take O^Z times, where both O and Z are large numbers. Which means that a person may take even his lifetime to complete the search of Ibnu Katheer, sequentially.

[125] A bibliographical reference is an example of citation networks. The Web of Science by Clarivent is a well-known example of citation networks. https://clarivate.com/webofsciencegroup/solutions/web-of-science/

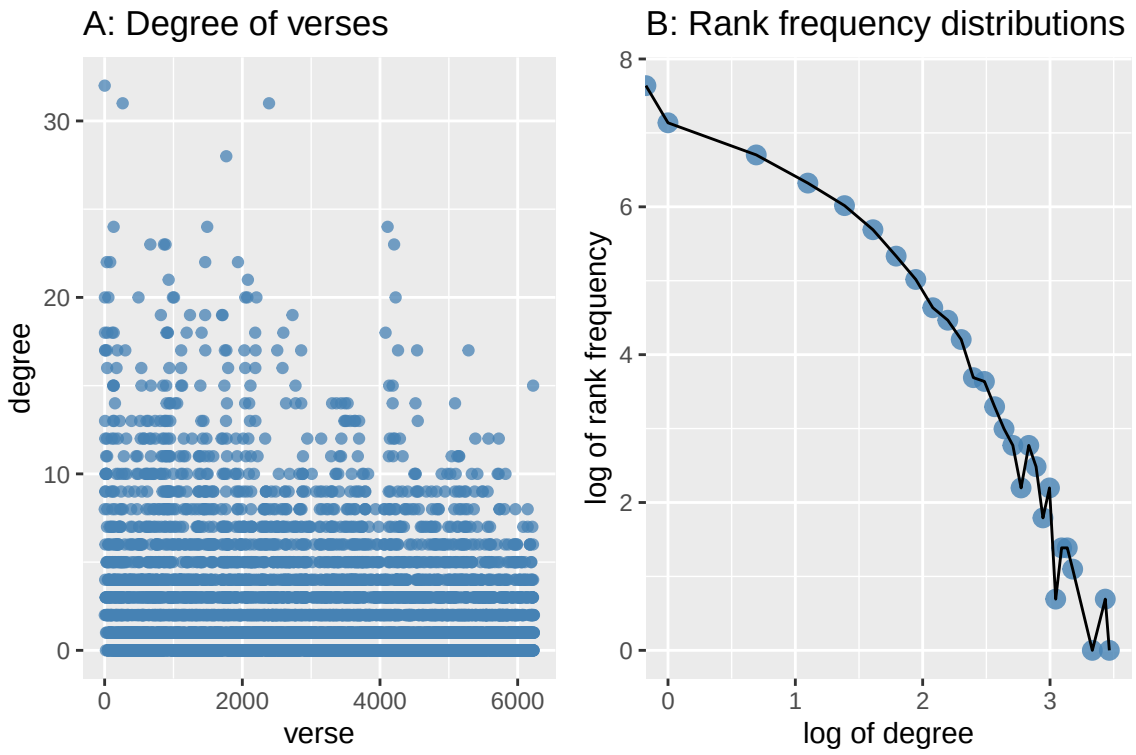


Figure 215: Plot of the Katheer Graph degree

What we have is a massively dense citation network. Every verse is an important message (literature) and is compactly present in the citation network. This fact is an amazing phenomenon! What this implies as well is that every verse (message) is important, not only on its own, but also within a peculiar way in the network. You can take out a verse randomly, it contains messages in many other verses within it. And if you take a group of connected verses, without one of its members, the message linkages are very unlikely to be broken.[126]

9.2.4 Katheer Graph network paths and traversals

Let us begin our discussion with the plot of the frequency distributions of *distance*, which is the measure of the number of "hops" in any path of the network. The plot is shown in Figure 216. This is an awesome discovery. The path distributions look like a normally distributed shape, with the average distance of 14.82. A verse on the network is about 15 links or hops away from any other verse, on average; or a message is never too far away from any other message. This is the "linear view" of it.

To appreciate the concept of distance better, let us ask the question how far is, say verse 6:25 (picked randomly), from any other verse in the network (picked randomly)? The plot in Figure 217 says that it is distributed like a normal distribution.[127]

We emphasize again that this observation is important, the network exhibits the small world property. A simple estimate of the small world property is measured by $log(N)/mean(d)$, which is 3.55. It means that in real terms, a verse is only about 4 links or steps away from "each other".[128]

[126]This is the same phenomenon in the Internet routers network; redundancies, alternate routings, etc., ensure that the network cannot fail. You can be connected to one of the routers and be accessible to the network, and yet if your router is broken (taken out), still the rest of the network operates smoothly.

[127]An important point to make is that it is a rare case when we see a well-behaved probability distribution for the paths of any network since most follow the Power Law structure.

[128]Note that distance is the measure of a verse from any other verse; whereas the small word property measure is "how far

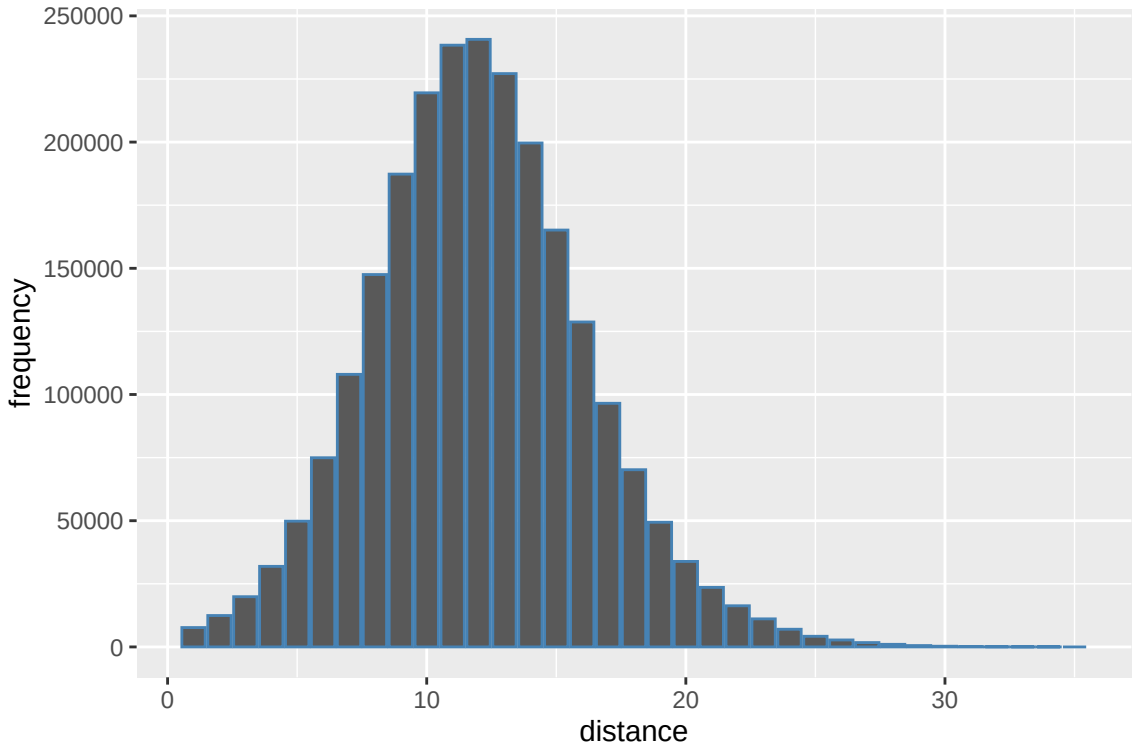


Figure 216: Plot of the Katheer Graph path lengths distributions

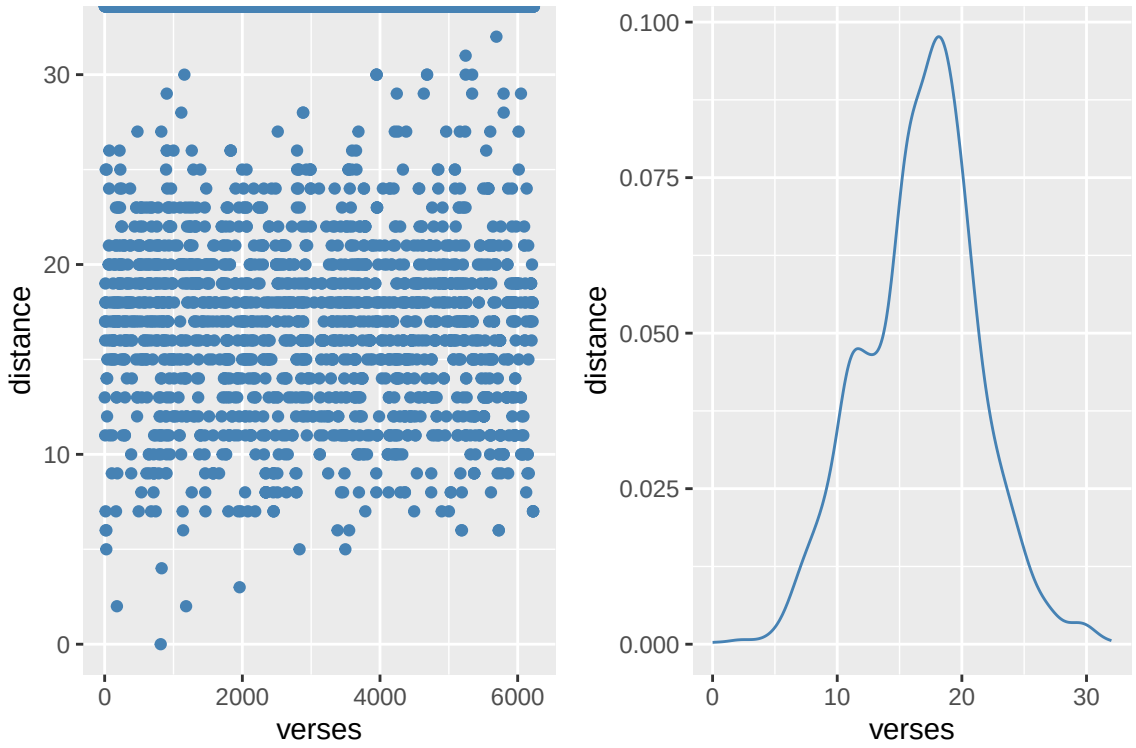


Figure 217: Verse 6:25 distances from other verses

This may imply that the messages within the verses are only a few steps away. We will demonstrate this observation later on in the chapter. Another way to say it is that even though the paths of interpretation can be quite long for some cases (can be as long as 51 steps), we do not need to go that "far back" in the traversals. Probably 4 steps is sufficient.

9.2.5 Network traversals using statistical properties

Analyzing network statistical properties is important in the sense that it describes the overall "behaviors" of the network. In network science, these behaviors reveal many "emergent" properties which are interesting and useful for explanatory purposes. We can plot and study various properties as we demonstrate next.

Some verses are "prestigious" or "highly influential" (ie., *eigencentrality* > 0.5) in interpreting other verses; and yet many of these verses are not necessarily contained within a grouping of verses (sub-networks) (i.e., *closenesscentrality* < 0.5). There are only a few verses that are both "prestigious" and at the same time have a high level of importance in their own groupings (i.e., *eigencentrality* > 0.5 and *closenesscentrality* > 0.5). Which are these verses? And how can we interpret these properties? If we dive deeper into the verses and the relations, we might have a better meaning of what they mean.

How about verses that are important in interpreting other verses as a carrier of the interpretation of verses to other verses? This is what's measured by *betweenness centrality*. Out of 6,000 over verses, more than 3,700 verses play this type of role as presented in the above plot. Again, this is an amazing observation. There are so many short verses in Al-Quran. And probably many of these short verses are actually links to other short (and also long) verses, and if taken together bring a new dimension of meanings to it.

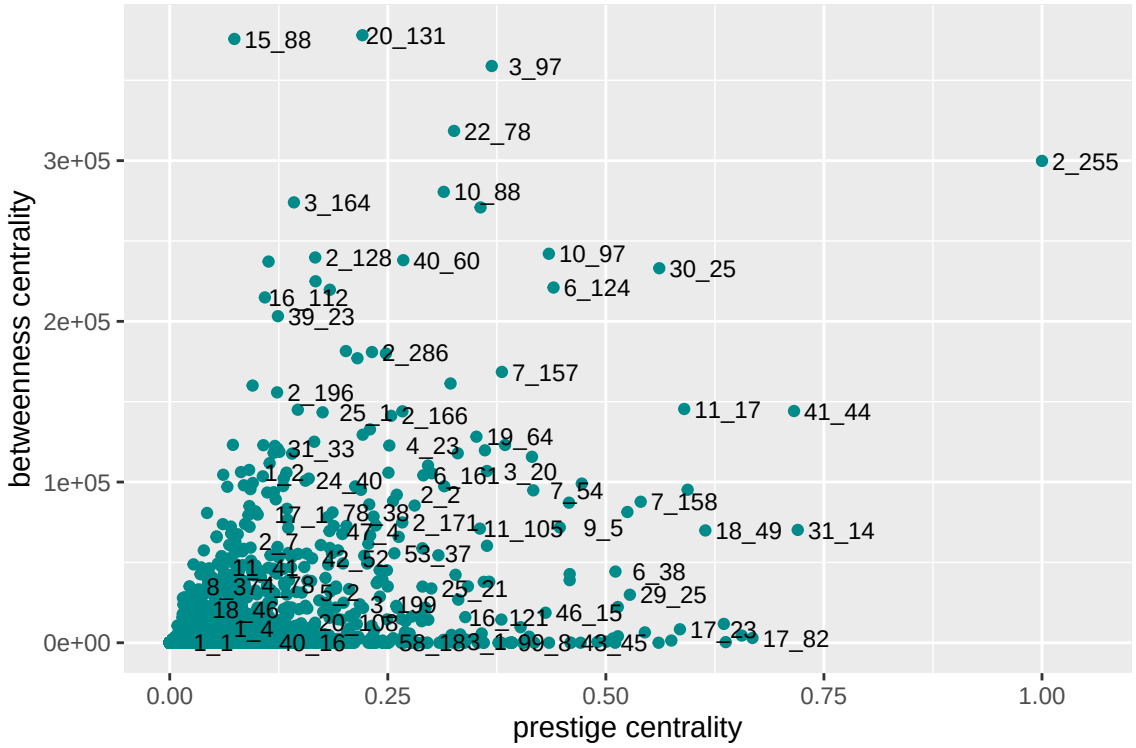


Figure 218: Centrality measures of verses in Ibnu Katheer: Prestige vs Betweeness

Verse 2__255

A plot of this is shown in Figure 218. We can observe clearly verse V2__255 is high, both in terms of "prestige" and "betweenness". What is verse 2:255? It is Ayah Al-Kursi. Comparing to the verses from

away" each verse truly is.

Al-Fatihah (opening Surah of Al-Quran), they rank lower on both counts (bottom left of the plot in Figure 218). Further checks reveal that indeed V2:255 has 16 in-degree (interpreted by 16 other verses) and 15 out-degree (interprets 16 other verses). But that is not the only fact, these 16 in-degree verses and 15 out-degree verses, in turn, have high in-degrees and out-degrees as well, for V2:255 to have high betweenness and prestige centrality. We will delve into the details of this in the latter part of the chapter.

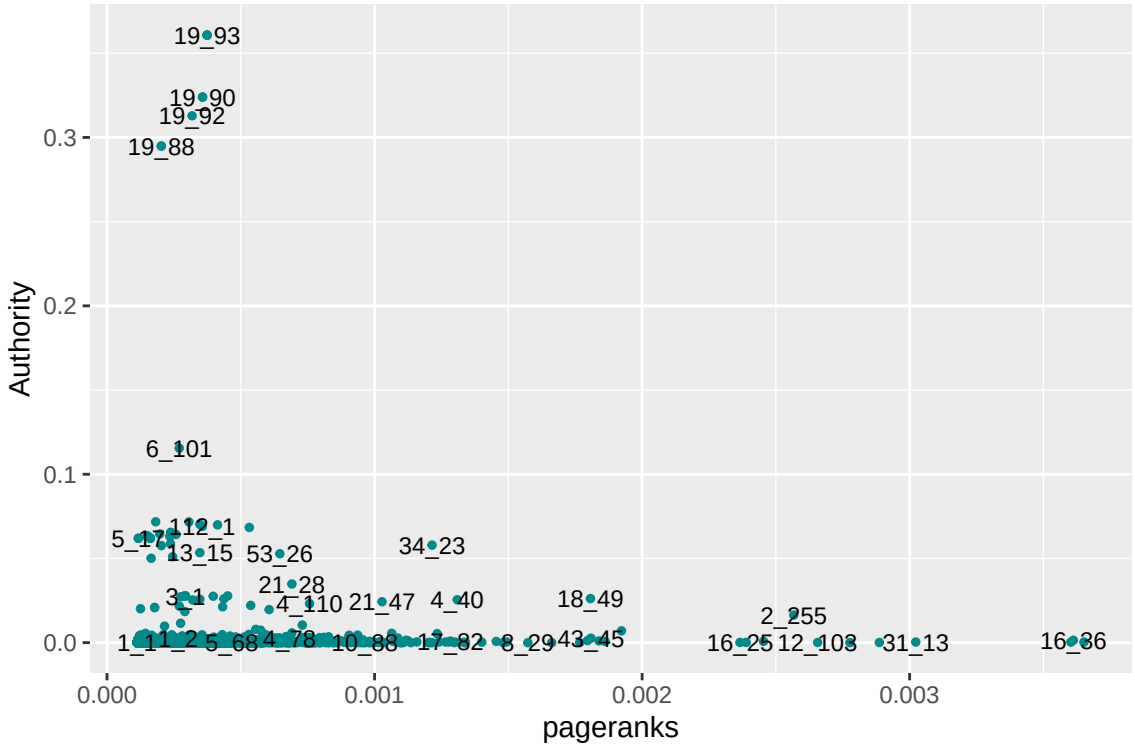


Figure 219: Centrality measures of verses in Ibnu Katheer: Pageranks vs Authority

Al-Rahman

Figure 219 compares the *Authority* and *pageranks* for the network. What can we learn from it? The group of verses from 19:88-93 (Surah Maryam v88-93) are high in terms of *Authority*, and yet low in terms of *pageranks*. What is the content of these verses? It contains the name of Allah, "Al-Rahman", befitting such knowledge that Al-Rahman is the most "authoritative" name of Allah (measured by *Authority*) but is mentioned not as often (measured by *pageranks*).

The Messengers and the message

How about Verse 16:36 (An-Nahl, verse 36)? Why is its "mentioned" status among the highest? A further look into the verse reveals that it is about the messenger (Prophets) with the message: *"Worship Allah and shun false gods".*

There are countless ways and methods we can traverse the network using the network statistical properties as we have demonstrated above, depending on the contexts and objectives. What we demonstrate here is just a sampling of what's possible. The method of visualization is just one of the simple ways to present a pictorial view of the subject. Tracing these links and traversing from one verse to another may yield new meanings and interpretations. The network graph model is a really useful tool in analyzing the work of Ibnu Katheer.

9.3 Traversals in Surah Al-A'laa {#traversals-in-Surah-Al-A'laa}

In this section, we will demonstrate an example of how the Ibnu Katheer network can be used within a study of a specific Surah, namely Surah Al-A'laa (Surah no 87). It is a short Surah with known virtues as noted by Imam Ibnu Katheer himself.[129]

The first step we have to do is to "extract out" the Surah from the entire network. This is important since we cannot build the network for the Surah in isolation, without linking it to the entire network. We must attempt to understand the Surah within the total framework, instead of a localized framework. A first pass exercise (or linear search) will be to go through each verse of the Surah, and observe the annotations by Ibnu Katheer for each of the verses; which is fine. But how about the second level search, which are the verses being referred to, how are they being referred to other verses at their own level? And we move to the next level, the same question arises, until the end of the traversal process.

This is achieved simply by combining the verses which point to any verse in Surah Al-A'laa (i.e. $E_i > 87_{1:19}$); then combine all the edges pointing to these verses ($E_j > E_i$) and move back up to five steps. The combination of edges creates the Surah Al-A'laa graph.

There are 44 verses pointing to the 15 verses (level 1); then there are 29 other verses pointing to the 44 verses (level 2); then there are 54 verses pointing to the 29 verses (level 3); then there are 135 verses pointing to the 54 verses (level 4); then there are 228 verses pointing to the 135 verses (level 5). We can continue as far as we want until there are no nodes and edges to be added onto the network which may reach 51 levels (the network diameter of the entire Ibnu Kahtheer network). Up to level 5, we have 490 edges with 231 nodes (verses).

What we want to do next depends on our objective. Let us say our objective is to understand how did Ibnu Katheer interpret Surah Al-A'laa, then we have to go through a few steps of questions and answers, as we will do next.

9.3.1 Themes of Surah Al-A'laa

What is/are the main theme(s) of the Surah? And in which larger theme(s) is it in? To answer the question we stack the network to 5 levels deep and we tabulate the results from STM topic modeling for various levels in the table below:

Level	No of verses	STM Top topic
0	15	brings pasture scriptures lord name abraham former indeed moses avoid
1	34	life hereafter worldly enjoyment except o revealed fire muhammad purifies
2	65	men women one believing day therein let say fire death
3	109	allah say o life except death fire know worldly better
4	201	allah indeed say life people o among many believed except
5	331	allah indeed lord people say us upon believed o know

Based on the English translation usage of wordings in Saheeh, we may say that the theme for the Surah is about: *"Allah say upon us to believe that (there is) life after death, (and) there is hell fire; and what is revealed through Prophet Muhammad (saw), (that we need to) purify and except (avoid) worldly enjoyment; and the (message is) in the scriptures of Abraham and Moses".*

How to make up the meaning of this sentence is up to an interpreter of Al-Quran. In fact, we can construct the sentence even better if we have the proper language model to put the proper ranking of the words (i.e. probabilistically). This is a subject we have to exclude from the current work. But the point is, we do have all the "data" necessary from the model (if we assume that the model is working), to construct a more organized sentence, semantically, syntactically, and grammatically.

We do not wish to pursue this direction for now at least due to two reasons:

[129] http://www.recitequran.com/tafsir/en.ibn-kathir/87:14

a) we need to work with a language model of English which is suitable for the task;
b) we have to assume the accuracy of Saheeh's interpretation is in full congruency with Ibnu Katheer's exegesis.

We cannot confirm either.

9.3.2 Surah Al-A'laa network

How does the Surah Al-A'laa network look like with five inward-pointing levels? This is shown in Figure 220, which is obtained from *Gephi* in order to provide a much clearer view of the network. The verses from Surah Al-A'laa are spread all over the network, as they are present within its own sub-network of verses interpreting it. By way of modularity grouping (as explained in earlier chapters), we colored the various "clusters" distinctly. This is the same basis as *cluster_fastgreedy* or *cluster_louvain* for finding *cliques* in the network.

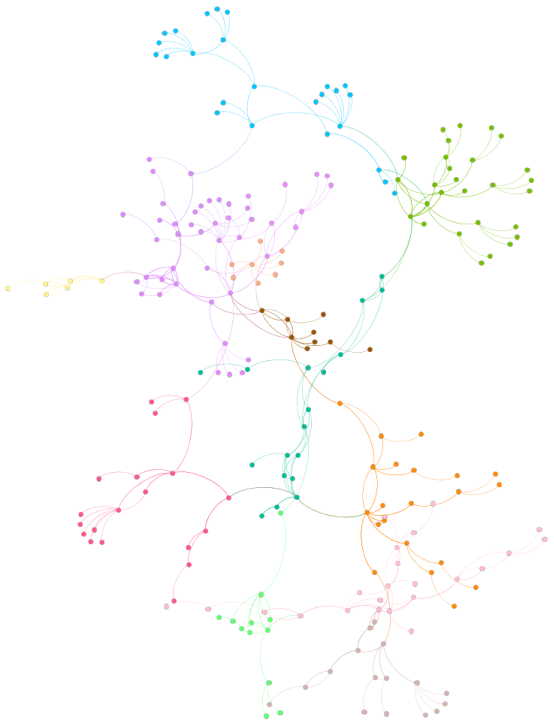


Figure 220: Verses links towards verses in Surah Al-A'laa, up to 5 levels

We create the graph using the *igraph* function and tabulate the summary statistics:

Statistics	Surah Al-A'laa network	Surah Al-Kahf	Surah Maryam
Nodes	231	NA	NA
Edges	490	NA	NA
Average degree	4.24	21.8	17.41
Diameter	8	7	7
Average Path Length	3.67	3.06	3.24
Transitivity	0.15	0.28	0.33

We can benchmark the results from the above table with similar tables in Chapter 7 for Surah Al-Kahf and

Surah Maryam. Note that the measures are for word co-occurrences network, which is a different framework than what we have here.

We calculate the network statistical measures and use STM to extract the topics in the top ten verses under each category. The results are tabulated below:

Measures	Topics results from STM on top ten verses of Surah Al-A'laa network
By Al-A'laa network top degree	allah, righteousness, death, gives, may, one, promise, true, angels, ask
By Al-A'laa network betweenness	allah, believing, errs, gives, guided, promise, true, righteousness, whoever, one

From the table, we can see the various word rankings in the network; depending on what perspective one wants to see, as an example, verses which are the most referred to (top degree), or the most referred to as in between references (betweenness). We can do the same for verses referred to less frequently but hold an important role (prestige), or by its high references using *pageranks* and numerous other measures (not shown here). Each of the settings has its own meaning as it represents one of the dimensions of "knowledge" through the rankings of words within the top topics.

9.3.3 View from the perspectives of the entire Ibnu Katheer network

We have not addressed other approaches like instead of looking "within the neighbors" of Surah Al-A'laa network, or we can traverse from the "neighbors outside" of Surah Al-A'laa, or within groupings of Surah such as the short Surahs. We can traverse outwardly or inwardly, depending on the objectives of learning.

Instead of using Al-A'laa network statistics to get the topics, we use the statistics obtained from the entire network of Ibnu Katheer Graph; and use STM to extract the topics. The results are tabulated below:

Measures	Topics extracted based on entire network measures
By entire network top degree	merciful, allah, entirely, especially, name, fear, guided, soul, aided, compensation
By entire network betweenness	whoever, call, earth, let, wills, indeed, lord, fire, like, truth

We can see that the order of the words changed (in the topic for the top ten verses), when we use the "original measures" from the entire network, instead of the measures from the "internal network" of Surah Al-A'laa. These changes are by no means trivial and have their own dimensions. A simple example may help: a person's status within his family members tells something about him, and at the same time his status in relation to all other people's status in their own families means something else. The topics of verses within a sub-network may tell us something, and if we measure relative to the entire network, those topics may rank differently.

How to interpret the results as presented, is beyond our current discussion for similar reasons that we made before: the STM model may suffer from small sample problems, and semantically we do not know the styles of language used in Saheeh translations. Therefore, we will leave the subject as it is and let the readers think for themselves as to what the results may imply. What we demonstrated are the methods and approaches for the extraction of information, via text topic modeling, relying on the network dimensions.

9.3.4 Summary

We can summarize for now that the possibilities of diving deeper into the "meaning" and "learning" from the Ibnu Katheer's interpretations of Al-Quran, by representing the verse-to-verse method of interpretations, are made easy when we transform the "knowledge" into "graphs". The network graph now allows the learner to

traverse through the layers of verses, links across verses, etc., which is very difficult to do otherwise. What we have shown here is still "raw" in the sense that, the extraction of meanings is based on the Saheeh English translations, and the semantic meaning is what we may be after.

Despite these shortcomings, we need to say that as far as the verses are concerned, the network we have shown and used for the traversals and explorations do not rely on the Arabic language at all. Furthermore, we do not rely on the comments made by Imam Ibnu Katheer in his exegesis. We rely solely on the data based on annotations made on the Tafseer.

9.4 Traversing verse 13 Surah Al-A'laa {#traversing-verse-13-Surah-Al-A'laa}

We choose a specific verse from a specific Surah to show how we can traverse from one verse to other verses in the network. This is slightly different from the previous section where we looked at a group of verses, namely the whole Surah Al-A'laa.

The verse 87:13 reads *"Neither dying therein nor living."* The phrase *"neither dying therein nor living"* is an important metaphor, which requires interpretation. Direct reading of Ibnu Katheer says that this verse is interpreted by verses 35:36 and 43:77. Verse 35:36 reads *"And for those who disbelieve will be the fire of Hell. [Death] is not decreed for them so they may die, nor will its torment be lightened for them. Thus do we recompense every ungrateful one."* and verse 43:77 reads *"And they will call,"O Malik, let your Lord put an end to us!" He will say, "Indeed, you will remain.""*

On the other hand, verse 87:13 is pointed towards two verses: 20:74 and 43:77. Notice that 43:77 is also pointed to 87:13, which is a circular loop, meaning that both interpret the other jointly. From the text of verse 43:77, it says about the Lord decreeing a judgement *"Indeed, you will remain","neither dying therein not living"*, and inversely, *"neither dying therein not living"*, *"Indeed, you will remain"*.

So far we have traversed through the first layer, which is a simulation of direct reading of Ibnu Katheer: read verse 87:13, look at the annotations, then read verse 35:36 and 43:77. We will only know that verse 87:13 is being referred to by verse 20:27 by reading verse 20:27 and finding its annotations, and similarly for verse 43:77. To go beyond this step, will involve repeating the process all over again from each verse mentioned. The process is tedious, lengthy, and maybe confusing, but that is what is exactly required without any shortcuts. We hope the readers appreciate the difficulty involved that we are trying to emphasize here.

Fortunately, this is all easily presented like a "map" using the network graph. We present this as an ego plot, in Figure 221. We can see that indeed, inward-looking, there are only three verses in the first layer (i.e. manual reading). However, beyond the first layer, there is a massive complex network that emanates from these three verses (35:36, 20:74, and 43:77). Behind 43:77 lies another massively complex network of its own going further into the layers, and similarly for 35:36.

By observing further we can see that verse 20:74 does not have such deeper networks emanating from it. Why? Apparently, the wordings of 20:74 are similar to 87:13, as verse 20:74 reads *"Indeed, whoever comes to his Lord as a criminal - indeed, for him is Hell; he will neither die therein nor live."* It is almost an exact word for word matching with verse 87:13.

How can a learner of Ibnu Katheer use this "map"? Firstly, he can easily draw the links between a verse to all the verses which point towards its interpretation. Secondly, the "map" presents the "message" or "knowledge" flows. As an example, reading verse 35:36 would likely confirm it is about the disbelievers and the torment of hellfire. The network around it probably presents a similar subject. Similarly, verse 43:77 is about the Lord's judgement on the disbelievers, a subject of the group of verses that relates to it, and so on.

Furthermore, based on our earlier point of "small word properties" of Katheer Graph, it is obviously clear where in the case of verse 87:13 network, that it is "only four or so steps away" from the rest of the verses. A way to understand this is just like your house is only four roads away from the main highway and the main highway defines the neighbourhood of your house. The interpretation of verse 87:13 can be obtained from the four hops exercise because beyond that the meaning is already apparent.

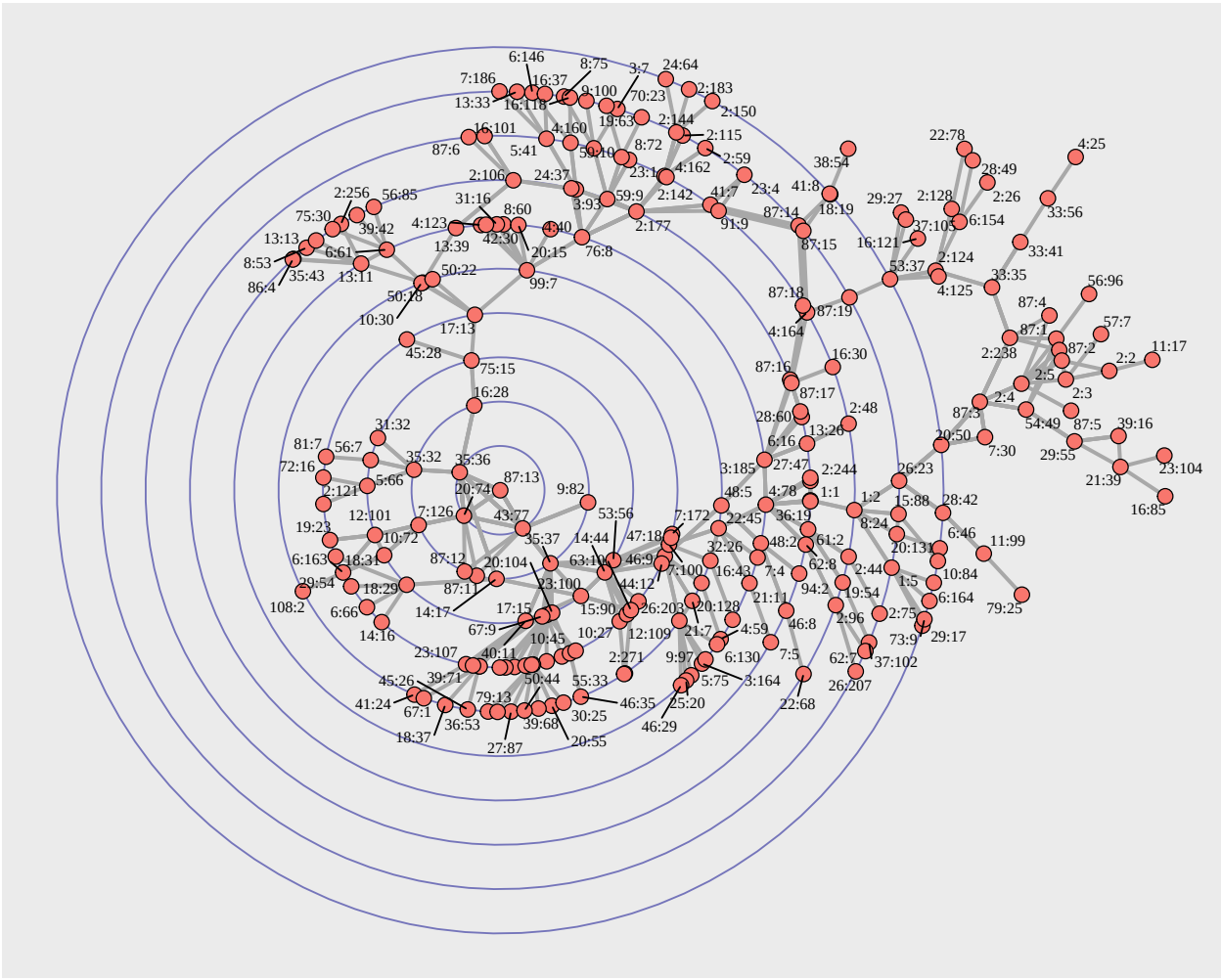


Figure 221: The verse map for V87:13

Additionally, even though a four layers analysis for verse 87:13 may suffice, it does not prevent a learner to see other systems "evolving". As an example, the ego network of 87:13 converges in the outer layer, layer 9, when the network starting from 35:36 and 43:77 > 35:37 meets again via verses 87:14 and 87:15 - which are direct neighbours of 87:13. What does this imply?

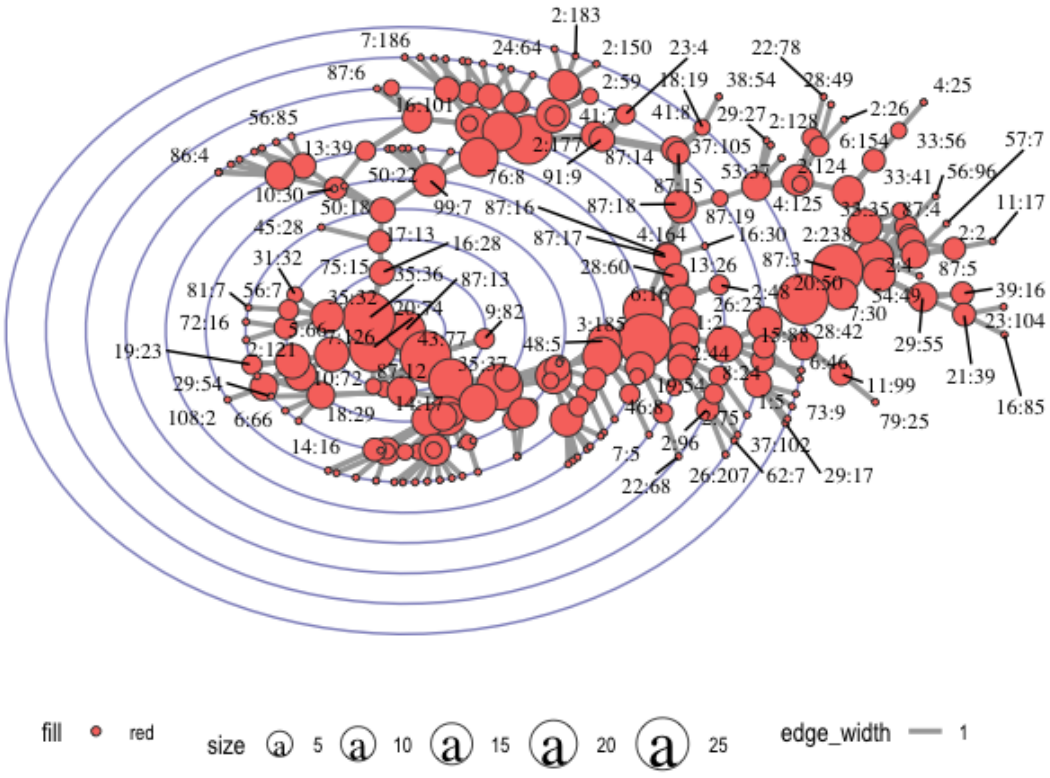


Figure 222: Verse 87:13 ego network directed with clusters

Now let us move further, to understand the network based on *betweenness centrality* measures (the concept of which we deliberated in earlier chapters). "Map" wise, we plot as in Figure 222 using the tools provided from earlier chapters. What can we say about it? The verses in the outer layers are organized in some peculiar settings - some verses serve as major connectors (as go-between) for other verses on the network. The large size nodes on the maps represent high connectivities compared to smaller size nodes. What messages it carries as go-betweenness is a subject a learner can go into.

Figure 223 uses *cluster_louvain()* as the clustering algorithm. We can see that the coloring schemes (labels) generated some groupings, which can be thought of as groups of distinct messages or themes. If a learner wants to understand them, then he should investigate each of the groupings and discover what are the themes which emerge as the significant elements within each. For a start, we can see that the center of the ego-network is no longer verse 87:13, instead, it is verse 17:15. Why this is the case? This is a classic "inversion" issue - that is when we invert the roles by importance within the entire network, another node becomes the "actual ego center".

Verse 17:15 reads: *"Whoever is guided is only guided for [the benefit of] his soul. And whoever errs only errs against it. And no bearer of burdens will bear the burden of another. And never would We punish until We sent a messenger."* It is about the subject of "guidance" and "no soul is burdened with the sin of others".

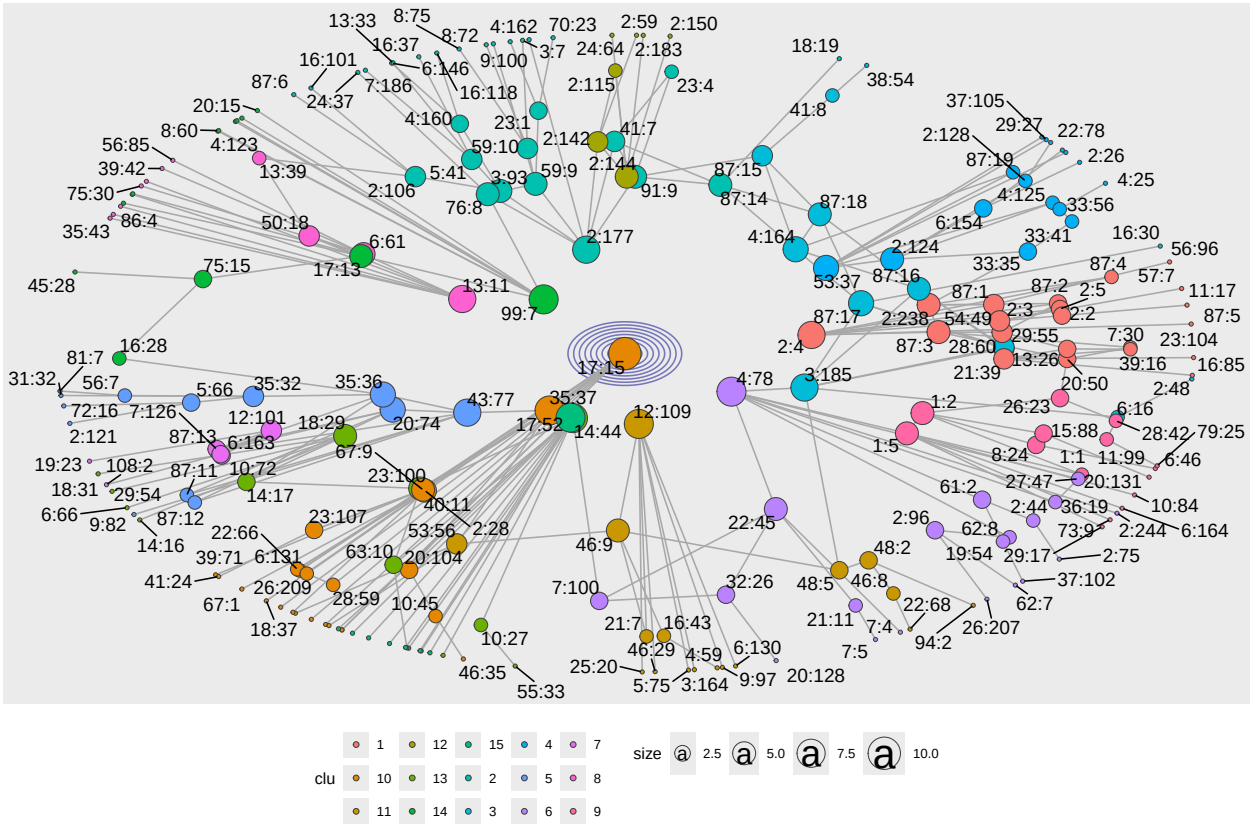


Figure 223: Inverse of verse 87:13 ego network

What this message implies should be of interest to a learner of the verse, within the context of verse 87:13, inversely.

Summary

How we can traverse the various paths, groupings, clusterings, just based on pointers and layouts as we have presented can be extensive and exhaustive. Thus far, we do not impose our own views on the interpretations of the verse, but instead, allow the "data to speak for itself". Furthermore, we also show how unsupervised learning models such as the STM model can be applied in combination with knowledge graphs. This is the power of knowledge graphs, and what we have demonstrated is about the tremendous amount of "knowledge" represented by the graph, just for a relatively short verse. We hope that the exercise convinces learners of Al-Quran of the strength and benefit of knowledge graph representations towards the knowledge retrieval process. We envisaged this to be a subject of paramount importance for Quran Analytics.

9.5 Traversals in verses 2:255 and 16:90

In this section, we will show another approach of graph traversals and clustering by joining the networks of two distant verses. The exercise is akin to independently looking at the interpretations of two separate verses together (reading different pages separated by quite a distance). We choose verses "2:255" (Ayat al-Kursi) and "16:90". "2:255" is well known for its virtues[130] and "16:90" is the command to be fair and just[131]. "2:255" describes many of the beautiful attributes of Allah (SWT). "16:90" summarizes the essence of Akhlaq or morals in Islam. "2:255" is about belief or faith (Iman) and "16:90" is about morals (Akhlaq).

We will take a different approach where 2:255 will be an "outwardly" moving network - that is which other verses 2:255 point to; and for 16:90, we look at which are the verses that interpret it (inwardly). Then we will combine the two networks together.

The "map from" 2:255 is in Figure 224. We can see that verse 2:255 interprets many other verses directly (nodes in layer 1), from layer 2 onwards, there are some groupings or clusterings of verses as we move outwardly, about four groupings. These groupings may well be "themes", "words" or any specific linkages - the subject of which will be known once we delve deeper into the verses and the linkages.

The "map towards" 16:90 is in Figure 225. The picture is a bit different, where we have many verses converging from many directions (hence sources) towards the center, verse 16:90. Possibly there are no unified themes involved.

We will combine these 2 ego graphs, where we have an outwardly 2:255 and inwardly 16:90. This is displayed in Figure 226. Now, we can see that verse 16:90 is in the outer layer, grouped under verse 4:48, connected via verse 31:13. Therefore, we can say that verse 16:90 (be fair and just) is part of the "larger" message of verse 2:255 (Ayah Al Kursi), through verse 31:13; and the message of which is contained in verse 31:13.

Now we invert the position, and make verse 16:90 the center and obtain the "map" in Figure 227. The inverted message, from verse 16:90, traversed through verse 31:13, which is the same verse as before (in non-inverted position). However, verse 20:211 becomes the only go-between to the remaining clusters of verse 2:255. The inverted message (of be fair and just) now pass through a "messenger" in-between to the larger message (Ayah Al-Kursi). Note that what we meant by inverted here is the orders are reversed.

Summary

The various traversals between the two verses which we have shown are another example of how we can use a knowledge graph representation in an unlimited number of ways. The most important point is the learner must set the objectives of learning and then follow a traversal path from the network that emanates from a selected verse. The original written text of Ibnu Katheer only provides the first degree or layer of connections inwardly. With a graph network, we can traverse inwardly or outwardly across many degrees, although "four hops" or "four degrees" will, on average, bring us back to the starting verse. From the path we have chosen,

[130] http://www.recitequran.com/tafsir/en.ibn-kathir/2:255
[131] http://www.recitequran.com/tafsir/en.ibn-kathir/16:90

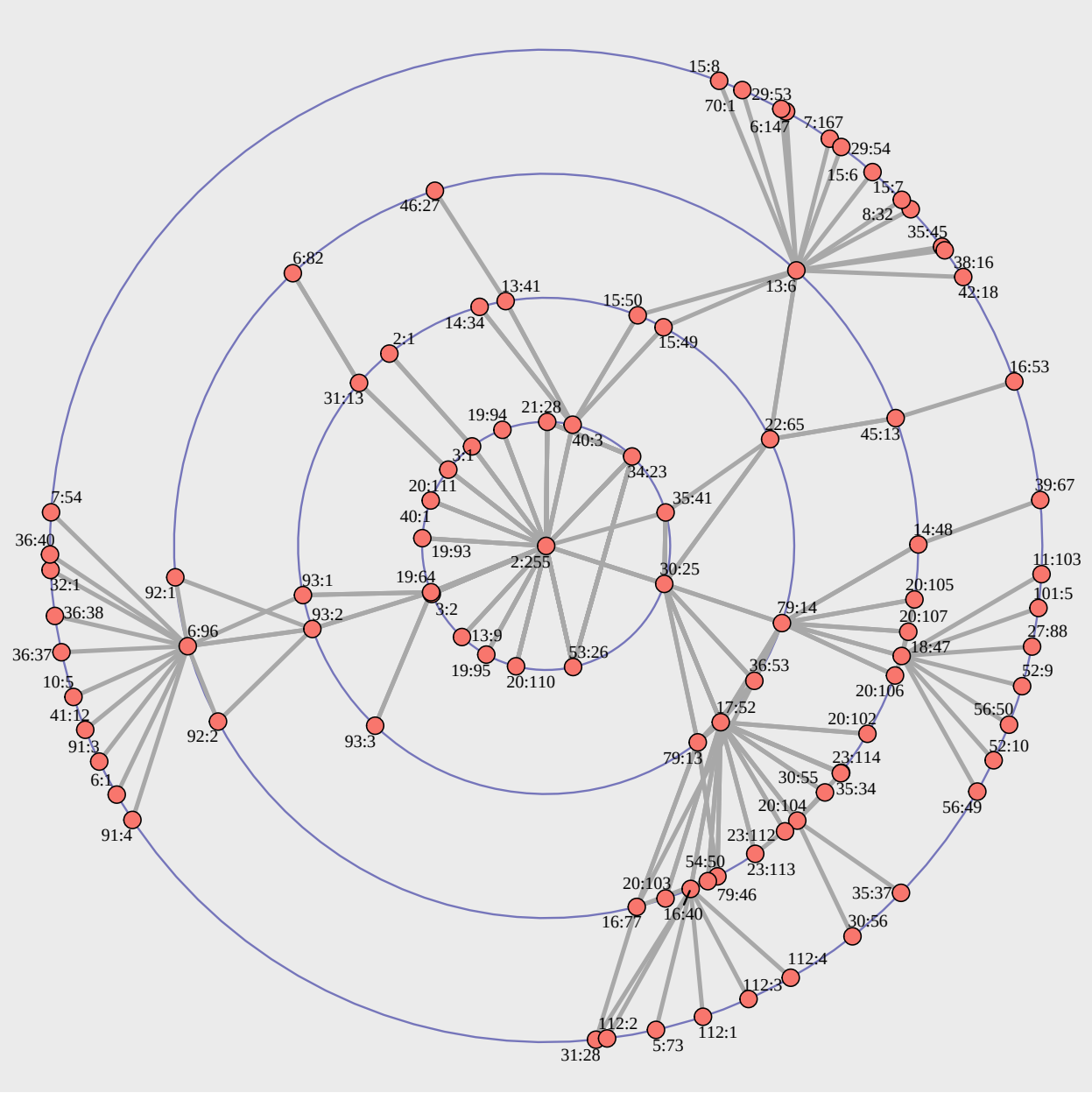


Figure 224: Verse 2:255 ego network outwardly

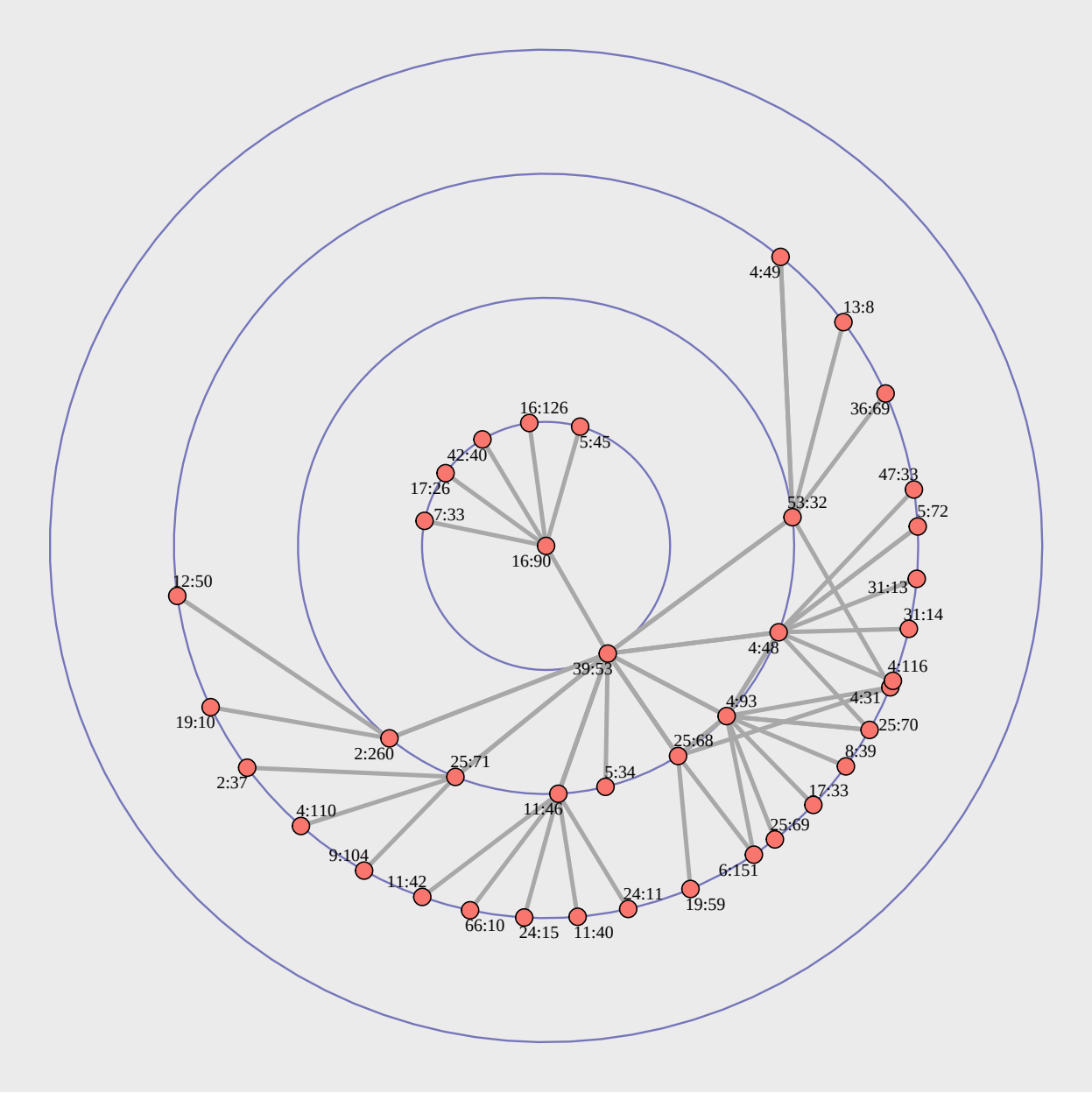


Figure 225: Verse 16:90 ego network inwardly

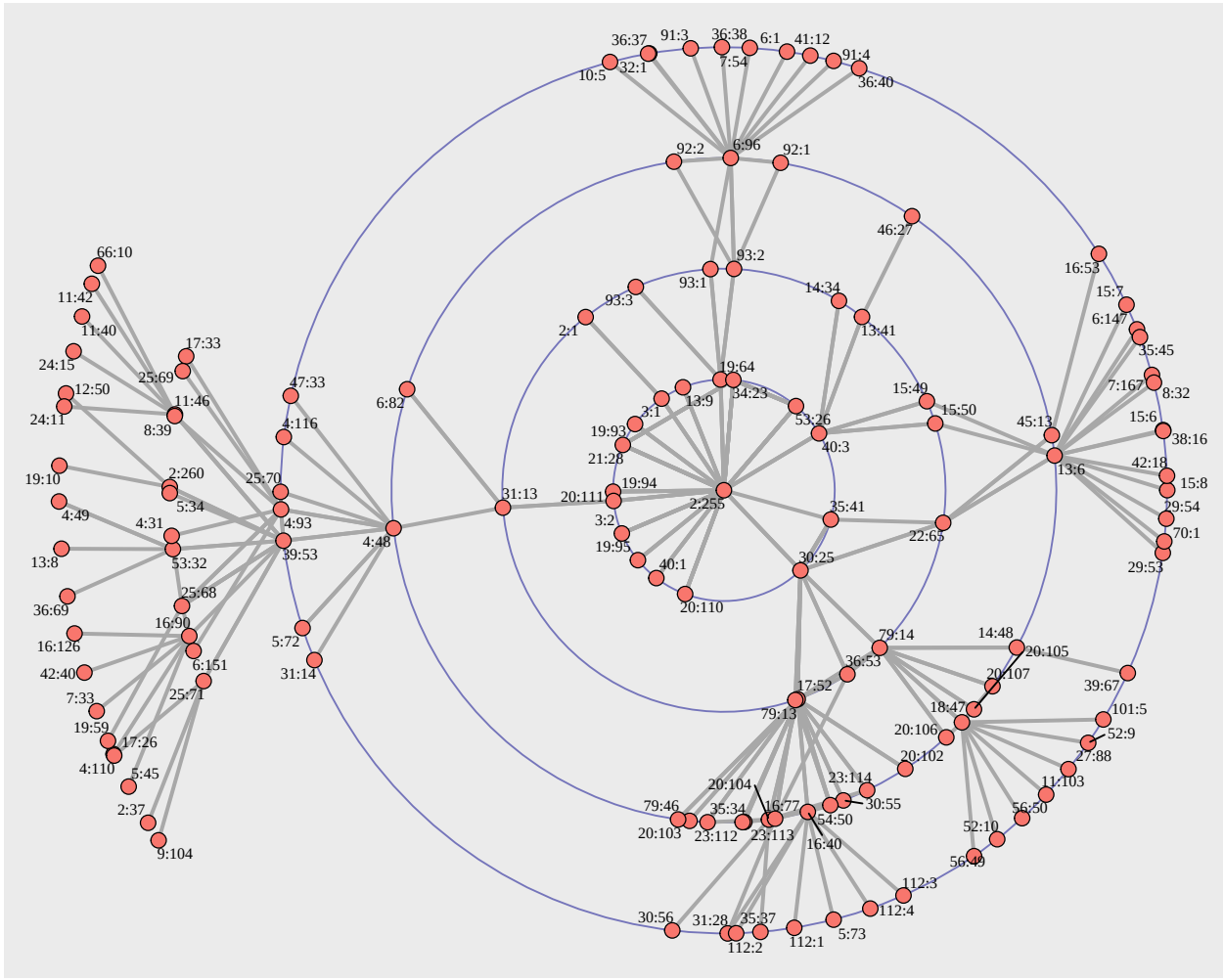


Figure 226: Outwardly map from verse 2:255 towards verse 16:90

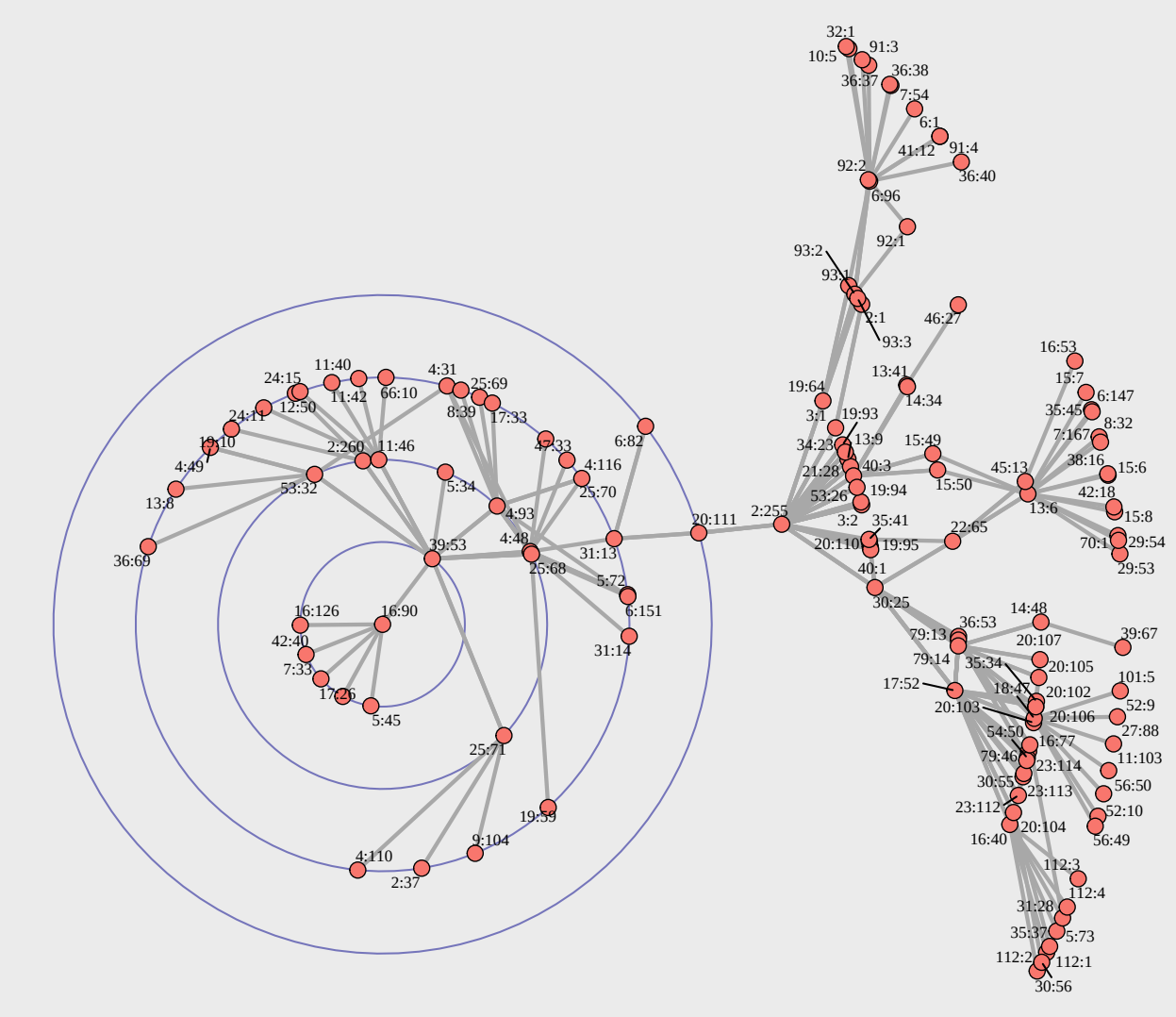


Figure 227: Verse 2:255 and 16:90 ego network union with 16:90 as focus node: first degree

we can then apply some of the "learning algorithms" that we have shown like STM, combined with some of the centrality properties of the network like betweenness.

9.6 Word cooccurrences from Katheer Graph

Word co-occurrences play a major role in language modeling in NLP, as we have explained in Chapters 5, 6, and 7. In fact, when Imam Ibnu Katheer made his commentaries, referring a verse to another, it is actually a referral based on parts or chunks of the verses. A chunk of a verse, interpreting another chunk. A chunk usually contains a few words. In some ways, we can think of it as similar to a synonym dictionary. The base of the chunk are words forming the sub-sentence. And when these chunks are "matched" together, we have the words to "co-occur" together. If we take this perspective, what will the word co-occurrence framework look like? Can we learn new things out of it?

Since the subject of the inquiry is a major one, we will not attempt to provide conclusive answers. Instead, we will work through some examples and demonstrate what are the possibilities offered by the methods we present in this book.

First and foremost, we need to note that the "normal" method of word co-occurrences by counting the frequencies of n-grams, as we have shown in Chapter 3, will not be applicable. Instead, we have to rely on the GloVe model of *word-2-vec* algorithms. The *word-2-vec* does not suffer from the problems of "localized" positions of n-grams; wherein n-grams, the exact position of the word in a sequence of sentences and documents matter. GloVe instead creates "global vectors" (as the name represents), where what we want to know is the position of a word in a global sense, and hence less on exact positional sense. Furthermore, GloVe is a dynamic model, where we can change and alter what we set as "global". The alterations depend on the objective and problem setting the researcher has.

We will apply the GloVe model and see how it works, with a note of caution. We have to rely on Saheeh's English translation, which confounds the analysis by an unknown degree since the translation methods and linguistic styles will enter into consideration when we want to interpret the results. Nevertheless, this is sufficient to demonstrate our objectives.

9.6.1 Setting the text data

We will again revisit and choose Surah Al-A'laa as our sample data. We will choose the network of verses up to five hops or levels inwardly and combine with five levels outwardly. In total, we have ten levels altogether. For demonstration, we will split the data into "inwardly" and "outwardly", and finally combine the data as a complete set.

9.6.2 What words co-occur together

First, we work with the "inwardly" network. We know from verse 87:13, the words "dying" and "living" are the keywords. Let us study the messages through word co-occurrences using the GloVe model as tabulated below:[132]

Inward network Surah Al-A'laa topical word co-occurrences

Topics	word cooccurrences
"allah"	allah, knowing, change, charitable, messenger, bring, initial
"lord"	lord, believed, spend, truth, relate, worlds, errs
"living"	living, deluded, taught, charity, command, water, previously
"dying"	dying, plotting, granted, waves, beware, averted, weep
"purify"	purify, terrify, garments, hell, error, treacherous, wrongs
"hell"	hell, confers, purify, question, cry, randomly, gathering

[132]The codes are in the Appendix.

Topics	word cooccurrences
combined	purify, lord, knowing, allah, hell, truth, garments

The results in the table above, say that for the words in the inward text networks, for example, *"allah, knowing, change, charitable, messenger, bring, initial"* which co-occurs with the word "allah", are the "inward" messages coming into Surah Al-A'laa. It is about *"knowing" and "allah"*. This is one way to look at it. Readers can see for themselves for the rest of the topics.

Outward network Surah Al-A'laa topical word co-occurrences

Topics	word cooccurrences
"allah"	allah, messenger, people, teach, enduring, realm, remember
"lord"	lord, truth, bedouins, unseen, promise, inclining, fear
"living"	living, accepted, stinginess, world, averting, pharaoh, invents
"dying"	dying, qiblah, excess, fall, duty, informed, covenant
"purify"	purify, striking, swallow, prayers, sound, fall, traveling
"hell"	hell, unlettered, fire, establisher, questioned, sons, withhold
combined	lord, allah, pardoned, purify, associate, dying, lying

The results in the table above, say that for the words in the outward text networks, for example, *"dying, qiblah, excess, fall, duty, informed, covenant"* which co-occurs with the word "dying", are the "outward" messages coming from Surah Al-A'laa. It is about *"dying, qiblah, excess, fall"*. This is one way to look at it. Readers can see for themselves the remaining topics.

Combined network Surah Al-A'laa topical word co-occurrences

Topics	word cooccurrences
"allah"	allah, remembrance, messenger, people, lord, knowing, life
"lord"	lord, truth, reason, allah, worship, likes, associate
"living"	living, prefer, preceded, succeeded, destined, dies, aid
"dying"	dying, delay, repaid, term, request, factions, disclose
"purify"	purify, house, keepers, forget, muslim, favor, shackles
"hell"	hell, fire, question, sleeping, regular, punishment, die
combined	allah, lord, dying, intend, worship, drop, signs

The results in the table above, say that for the words in the combined inward and outward text networks, for example, *"hell, fire, question, sleeping, regular, punishment, die"* which co-occurs with the word "hell", are the full traversed messages for Surah Al-A'laa. It is about *"hell, fire, question, sleeping"*. This is one way to look at it. The readers should be able to figure out themselves for the rest of the topics.

It is not our intention to use all of these exercises to **interpret** Surah Al-A'laa. What we show is just a methodology to extract information based on a defined objective. The objective is set based on the assumptions that words in a language tend to occur together, and jointly they provide semantical and ontological meaning to the subject of the texts.

9.7 Summary

This chapter glues together many concepts, ideas, and methods of NLP tasks, that we have introduced in the earlier chapters based on **R** and the selected packages.

Programming in **R** is not too hard and the credit mainly goes to the many developers of packages related to NLP. The *igraph*, *ggraph*, and the plotting functions of *ggplot2* are of extreme benefit when we deal with graph objects and their statistical calculations, as well as graph algorithms. For someone with moderate programming experience, a few months of training will get him to a sufficient level to run all the codes we have used here. Someone proficient in any programming language should take less than a month.

Knowledge graphs as representations of a network of information such as the Ibnu Katheer exegesis show both the brilliance and knowledge of Imam Ibnu Katheer and the easiness for us to understand his brilliance and knowledge.

The work we have shown here for Imam Ibnu Katheer Graph for "Tafseer Al-Quran bi Al-Quran" barely scratches the surface. Our intention is to demonstrate the knowledge graph for Quran Analytics and use Tafseer Ibnu Katheer as a sample. It is our view that the many possible further avenues of research are way beyond what we have worked out. Albeit, our purpose was to introduce, rather than to conclude.

For our benefit and the readers, we list what we think are the areas of further research based on some of the results of this chapter:

1. The full work of Katheer Graph needs to be analyzed using the Arabic language as its source of text. Note that in the work of this chapter, we use the English translations only when we want to provide some textual results based on the verses groupings. The main results, which are "numbers and pointers" do not assume any language. It is just data as is. Everything that we have done, can be performed in Arabic without any problem. Even the GloVe model is language-independent. This is what we meant by non-parametric models of NLP where the dependency on any pre-built model is avoided.

2. There are more methods of discovery for the Katheer Graph which we have not tried. Exploring these methods certainly involves advanced research that is suitable for Ph.D. thesis and research papers. The brilliance and knowledge of Imam Ibnu Katheer of Al-Quran and his methodologies are hidden gems. The fastest method to do this is to apply knowledge graphs on a larger scale; that is to convert and annotate at the "words" level, on top, and above the verse level. This can be combined together with his annotations of Al-Hadith, and his commentaries. Only then, we will have a full appreciation of the exegesis work by Imam Ibnu Katheer.

3. Utilization of unsupervised learning models for NLP tasks, like GloVe algorithms, deep learning, and Graph NLP, is still among the fresh fields of knowledge and discovery. Applications to Quran Analytics using these new tools are scarce. We envisage that combining Knowledge Graph, Graph Theory, Systems Theory, Unsupervised Learning NLP modeling, combined with Artificial Intelligence and Machine Learning computing algorithms will be among the potent and powerful tools for studies of Al-Quran and Islamic knowledge and sciences.

Quran Analytics will be the starting point for us to develop and implement full-scale knowledge extraction systems from classical sources and writings on Al-Quran. We feel that the day for realizing this is near.

9.8 Further readings

Barabasi, A.-L. (2016). *Network Science*. Cambridge University Press, Cambridge, Massachussetts

Blumauer, A. (2016). *From taxonomies over ontologies to knowledge graphs.* Available at https://semantic-web.com/2014/07/15/from-taxonomies-overontologies-to-knowledge-graphs/ (2021/02/16). Paulheim, H. (2016). *Knowledge graph refinement: A survey of approaches and evaluation methods.* Semantic Web Journal, Preprint:1–20.

Reinanda, R.,Meij,E., and de Rijke, M.(2020). *Knowledge graphs: An information retrieval perspective.* Foundations and Trends in Information Retrieval, 14(4):289–444.

Singhal, A. (2012). *Introducing the knowledge graph: things, not strings.* Available at https://blog.google/products/search/introducing-knowledge-graphthings-not/ (2021/02/16).

Appendix

Codes for Chapter 9

```
### Source for Chapter 9 codes
## Wan M Hasni and Azman Hussin
## This codes will be run prior to running the Rmd files for Chapter 9

## load quran sahih
kvertices <- quran_en_sahih %>%
  select(ayah_title, surah_id, ayah, ayah_id,
         surah_title_en, revelation_type, text)

## load ibnu_kathir and process
kathir <- read_csv("data/ibn_kathir_similarity.csv",
                   col_types = cols(ID = col_integer(),
                                    Source_Sura = col_integer(),
                                    Source_Verse = col_integer(),
                                    Target_verse = col_integer(),
                                    common_roots = col_integer(),
                                    relevance_degree = col_integer(),
                                    target_sura = col_integer()))
kathir <- kathir %>%
  rename(SourceSura = Source_Sura, SourceAyah = Source_Verse,
         TargetSura = target_sura, TargetAyah = Target_verse)

kathir <- kathir %>%
  drop_na() %>%
  mutate(from = str_c(as.character(SourceSura),
                      as.character(SourceAyah),sep = ":"),
         to = str_c(as.character(TargetSura),
                    as.character(TargetAyah),sep = ":")) %>%
  rename(weight = relevance_degree)

## create edge-list
kedges <- kathir %>%
                select(from, to, SourceSura, SourceAyah, TargetSura,
                       TargetAyah, common_roots, weight)

kg = graph_from_data_frame(kedges, directed = TRUE, vertices = kvertices)

## read files for nodes and attributes from gephi output
ik_gephi = read_csv("data/ik1_nodes.csv")

## Katheer Graph Network stats
kg_degree = data.frame("verse"=1:6236, "degree"= degree(kg))
fig901a = kg_degree %>%  ggplot() +
  geom_point(aes(x=verse,y=degree), color = "steelblue", alpha = 0.75) +
  labs(title="A: Degree of verses")

kg_degree_ranked = kg_degree[rev(order(kg_degree$degree)),]

fig901b = kg_degree_ranked %>% group_by(degree) %>% count() %>%
    ggplot(aes(x=log(degree), y=log(n))) +
    geom_point(color="steelblue", size = 3, alpha = 0.8, show.legend = FALSE) +
    geom_path() +
    labs(title = "B: Rank frequency distributions",
         x = "log of degree",
         y = "log of rank frequency")

kg_dist_table = distance_table(kg)

fig902 = ggplot() + geom_col(aes(x = 1:35,y=kg_dist_table$res), color = "steelblue") +
  labs(x = "distance", y = "frequency")

kgd = distances(kg, v = V(kg), to = V(kg), mode = "in")

fig903a = data.frame("x" = 1:6236, "y" = kgd[,"6:25"]) %>%
  ggplot() + geom_point(aes(x,y), color="steelblue") +
  labs(x = "verses", y = "distance")

fig903b = data.frame("x" = 1:6236, "y" = kgd[,"6:25"]) %>%
  ggplot() + geom_density(aes(y), color="steelblue") +
  labs(x = "verses", y = "distance")

## fig903: Centrality measures of verses in Ibnu Katheer: Prestige vs Betweeness
fig903c = ik_gephi %>%
                ggplot(aes(x = eigencentrality, y = betweenesscentrality)) +
                    geom_point(color = "cyan4") +
```

```
                    geom_text(label=ik_gephi$Id, size = 3,
                              nudge_x = 0.05, nudge_y = 0.05,
                              check_overlap = T) +
                    labs(x = "prestige centrality", y = "betweenness centrality")
## fig904: Centrality measures of verses in Ibnu Katheer: Pageranks vs Authority
fig904 = ik_gephi %>% ggplot( aes(x=pageranks, y=Authority)) +
                      geom_point(size = 1, color = "cyan4") +
                      geom_text( aes(label=ik_gephi$Id), size = 3,
                                 check_overlap = T) +
                      labs(x = "pageranks", y = "Authority")

## TRAVERSALS IN SURAH AL-A'ALAA

## create the df for edges
surah87_l1 = kedges %>% filter(str_detect(to,"87:")) %>% select(from,to)
surah87_l2 = kedges %>% filter(to %in% surah87_l1$from) %>% select(from,to)
surah87_l3 = kedges %>% filter(to %in% surah87_l2$from) %>% select(from,to)
surah87_l4 = kedges %>% filter(to %in% surah87_l3$from) %>% select(from,to)
surah87_l5 = kedges %>% filter(to %in% surah87_l4$from) %>% select(from,to)

## Topic modeling within the verses

# 1. create a function to be used
## Use STM model for topics, get Topics as output
surah87_topics = function(txt87){
                    dfm_txt87 = quanteda::tokens(txt87,"word",remove_punct = T,
                                remove_symbols=T,
                                remove_numbers=T) %>%
                    quanteda::tokens_tolower() %>%
                    quanteda::tokens_remove(quanteda::stopwords("en")) %>%
                    quanteda::dfm()
                    stmdfm_txt87 = quanteda::convert(dfm_txt87, to = "stm")
                    stm_txt87 = stm::stm(stmdfm_txt87$documents,
                                         stmdfm_txt87$vocab, K = 3, verbose = F,
                                         data = stmdfm_txt87$meta,
                                         init.type = "Spectral")
                    stm::labelTopics(stm_txt87, c(1), n = 10)}

## 2. Run using the function

txt87 = quran_en_sahih %>% filter(ayah_title %in% surah87_l1$to) %>% pull(text)
topics_l0 = paste(surah87_topics(txt87)[["prob"]][1,], collapse = " ")

surah87n_l1 = surah87_l1
txt87 = quran_en_sahih %>% filter(ayah_title %in%
                                  c(surah87n_l1$from,surah87n_l1$to)) %>% pull(text)
topics_l1 = paste(surah87_topics(txt87)[["prob"]][1,], collapse = " ")

surah87n_l2 = bind_rows(surah87_l1,surah87_l2)
txt87 = quran_en_sahih %>% filter(ayah_title %in%
                                  c(surah87n_l2$from,surah87n_l2$to)) %>% pull(text)
topics_l2 = paste(surah87_topics(txt87)[["prob"]][1,], collapse = " ")

surah87n_l3 = bind_rows(surah87_l1,surah87_l2,surah87_l3)
txt87 = quran_en_sahih %>% filter(ayah_title %in%
                                  c(surah87n_l3$from,surah87n_l3$to)) %>% pull(text)
topics_l3 = paste(surah87_topics(txt87)[["prob"]][1,], collapse = " ")

surah87n_l4 = bind_rows(surah87_l1,surah87_l2,surah87_l3,surah87_l4)
txt87 = quran_en_sahih %>% filter(ayah_title %in%
                                  c(surah87n_l4$from,surah87n_l4$to)) %>% pull(text)
topics_l4 = paste(surah87_topics(txt87)[["prob"]][1,], collapse = " ")

surah87n_l5 = bind_rows(surah87_l1,surah87_l2,surah87_l3,surah87_l4,surah87_l5)
txt87 = quran_en_sahih %>% filter(ayah_title %in%
                                  c(surah87n_l5$from,surah87n_l5$to)) %>% pull(text)
topics_l5 = paste(surah87_topics(txt87)[["prob"]][1,], collapse = " ")

## outputs are: topics_l1,..,l5

## 3. create igraph from edge list created above

surah87_net = graph_from_edgelist(as.matrix(surah87n_l5), directed = TRUE)

## then create measures to be quoted inside Rmd texts
ayat_topdegree = degree(surah87_net)
ayat_topdegree = ayat_topdegree[rev(order(ayat_topdegree))]
ayat_topdegree = names(ayat_topdegree[1:10])

ayat_topbtwn = betweenness(surah87_net)
```

```
ayat_topbtwn = ayat_topbtwn[rev(order(ayat_topbtwn))]
ayat_topbtwn = names(ayat_topbtwn[1:10])

ayat_topevcent = eigen_centrality(surah87_net)$vector
ayat_topevcent = ayat_topevcent[rev(order(ayat_topevcent))]
ayat_topevcent = names(ayat_topevcent[1:10])


## make some changes to the data, replace _ with :
ik_gephi$idnew =str_replace(ik_gephi$Id, "_",":") # change the separator

## create more variables to use for the tables in Rmd document
ayat_ovtdg = ik_gephi %>% filter(idnew %in% c(surah87n_l5$from,surah87n_l5$to)) %>%
                          select(idnew, Degree,eigencentrality,betweenesscentrality)
ayatdg = ayat_ovtdg[rev(order(ayat_ovtdg$Degree)),]
ayatbw = ayat_ovtdg[rev(order(ayat_ovtdg$betweenesscentrality)),]
ayatev = ayat_ovtdg[rev(order(ayat_ovtdg$eigencentrality)),]

## topics for top 10 grouped by Surah 87 net measures

topd10txt = quran_en_sahih %>% filter(ayah_title %in% ayat_topdegree) %>% pull(text)
td10 = surah87_topics(topd10txt)
topb10txt = quran_en_sahih %>% filter(ayah_title %in% ayat_topbtwn) %>% pull(text)
tb10 = surah87_topics(topb10txt)

ayatd10txt = quran_en_sahih %>% filter(ayah_title %in% ayatdg$idnew[1:10]) %>% pull(text)
ad10 = surah87_topics(ayatd10txt)
ayatb10txt = quran_en_sahih %>% filter(ayah_title %in% ayatbw$idnew[1:10]) %>% pull(text)
ab10 = surah87_topics(ayatb10txt)
## outputs used inside the Rmd text

########################
## Traversing verse 13 Surah Al-A'laa

## capture the edges for Surah Al-A'laa
vs8713 = kvertices %>% filter(ayah_title=="87:13") %>% pull(text)
vs3536 = kvertices %>% filter(ayah_title=="35:36") %>% pull(text)
vs4377 = kvertices %>% filter(ayah_title=="43:77") %>% pull(text)
vs2074 = kvertices %>% filter(ayah_title=="20:74") %>% pull(text)
vs1715 = kvertices %>% filter(ayah_title=="17:15") %>% pull(text)

# Making and plotting ego graphs
verse <- which(V(surah87_net)$name=="87:13")
v_name = V(surah87_net)$name
fig_egos87 = ggraph(surah87_net, layout = "focus", focus = verse) +
  draw_circle(col = "darkblue", use = "focus",max.circle = 10) +
  geom_edge_link0(aes(edge_width = 1),edge_colour = "grey66") +
  geom_node_point(aes(fill="red", size=3), shape = 21) +
  geom_node_text(size = 3,label = v_name,
                 family = "serif", repel=T) +
  scale_edge_width_continuous(range = c(0.1,2.0)) +
  scale_size_continuous(range = c(1,5)) +
  coord_fixed() +
  theme_graph() +
  theme(legend.position = "none")

## Clustering measures
cld <- cluster_edge_betweenness(surah87_net, weights = NULL)
mem <- membership(cld)
com <- communities(cld)

# Verse 87:13 ego network directed with clusters

## Figure 9.7 "map based brtweeness centrality
my_palette <- c("red", "blue", "green", "gold", "cyan4", "deeppink", "tomato", "yellow")
cld <- clusters(surah87_net)
V(surah87_net)$clu <- as.character(cld$membership)
V(surah87_net)$size <- graph.strength(surah87_net)

verse <- which(V(surah87_net)$name=="87:13")
v_name = V(surah87_net)$name
fig907x = ggraph(surah87_net,layout = "focus", focus = verse) +
    draw_circle(col = "darkblue", use = "focus",max.circle = 10) +
    geom_edge_link0(aes(edge_width=1), edge_color="grey66") +
    geom_node_point(aes(fill="red", size=size), shape=21, col="grey25") +
    geom_node_text(aes(size=2.5, label=v_name), family = "serif", repel=T) +
    scale_edge_width_continuous(range=c(0.1, 2.0)) +
    scale_size_continuous(range=c(1,10)) +
    # scale_fill_manual(values=my_palette) +
    theme_graph(title_size = 16, subtitle_size = 14) +
```

```
    theme(legend.position = "bottom")

## Verse 87:13 undirected ego network with centrality layout, clusters strength"}
kgu <- simplify(as.undirected(surah87_net))
clu <- cluster_louvain(kgu)
V(kgu)$clu <- as.character(clu$membership)
V(kgu)$size <- graph.strength(kgu)

fig909 = ggraph(kgu, "centrality", cent=graph.strength(kgu)) +
  draw_circle(col = "darkblue", use = "focus", max.circle = 10) +
  geom_edge_link0(edge_color="grey66") +
  geom_node_point(aes(fill=clu, size=size), shape=21, col="grey25") +
  geom_node_text(aes(size=2.5,label=name), repel=T) +
  scale_edge_width_continuous(range=c(0.1,2.0)) +
  scale_size_continuous(range=c(1,10)) +
  theme_graph(title_size = 16, subtitle_size = 14) +
    theme(legend.position = "bottom")

## Combining verses up to 4 layers outwards

v2255_l1 = kedges %>% filter(str_detect(from,"2:255")) %>% select(from,to)
v2255_l2 = kedges %>% filter(from %in% v2255_l1$to) %>% select(from,to)
v2255_l3 = kedges %>% filter(from %in% v2255_l2$to) %>% select(from,to)
v2255_l4 = kedges %>% filter(from %in% v2255_l3$to) %>% select(from,to)
#v2255_l5 = kedges %>% filter(to %in% v2255_l4$from) %>% select(from,to)

v2255 = bind_rows(v2255_l1,v2255_l2,v2255_l3,v2255_l4)
v2255_net = graph_from_edgelist(as.matrix(v2255), directed = TRUE)

v1690_l1 = kedges %>% filter(str_detect(from,"16:90")) %>% select(from,to)
v1690_l2 = kedges %>% filter(to %in% v1690_l1$from) %>% select(from,to)
v1690_l3 = kedges %>% filter(to %in% v1690_l2$from) %>% select(from,to)
v1690_l4 = kedges %>% filter(to %in% v1690_l3$from) %>% select(from,to)
v1690_l5 = kedges %>% filter(from %in% v1690_l4$to) %>% select(from,to)

v1690 = bind_rows(v1690_l1,v1690_l2,v1690_l3,v1690_l4,v1690_l5)
v1690_net = graph_from_edgelist(as.matrix(v1690), directed = TRUE)

verse <- which(V(v2255_net)$name=="2:255")
v_name = V(v2255_net)$name
fig910 = ggraph(v2255_net, layout = "focus", focus = verse) +
  draw_circle(col = "darkblue", use = "focus",max.circle = 4) +
  geom_edge_link0(aes(edge_width = 1),edge_colour = "grey66") +
  geom_node_point(aes(fill="red", size=3), shape = 21) +
  geom_node_text(size = 3,label = v_name,
                 family = "serif", repel=T) +
  scale_edge_width_continuous(range = c(0.1,2.0)) +
  scale_size_continuous(range = c(1,5)) +
  coord_fixed() +
  theme_graph() +
  theme(legend.position = "none")

verse <- which(V(v1690_net)$name=="16:90")
v_name = V(v1690_net)$name
fig911 = ggraph(v1690_net, layout = "focus", focus = verse) +
  draw_circle(col = "darkblue", use = "focus",max.circle = 4) +
  geom_edge_link0(aes(edge_width = 1),edge_colour = "grey66") +
  geom_node_point(aes(fill="red", size=3), shape = 21) +
  geom_node_text(size = 3,label = v_name,
                 family = "serif", repel=T) +
  scale_edge_width_continuous(range = c(0.1,2.0)) +
  scale_size_continuous(range = c(1,5)) +
  coord_fixed() +
  theme_graph() +
  theme(legend.position = "none")

v_combined = bind_rows(v2255,v1690)
vcomb_net = graph_from_edgelist(as.matrix(v_combined), directed = TRUE)

vcomb_net = graph.union(v2255_net,v1690_net)
verse <- which(V(vcomb_net)$name=="2:255")
v_name = V(vcomb_net)$name
fig912 = ggraph(vcomb_net, layout = "focus", focus = verse) +
  draw_circle(col = "darkblue", use = "focus",max.circle = 4) +
  geom_edge_link0(aes(edge_width = 1),edge_colour = "grey66") +
  geom_node_point(aes(fill="red", size=3), shape = 21) +
  geom_node_text(size = 3,label = v_name,
                 family = "serif", repel=T) +
  scale_edge_width_continuous(range = c(0.1,2.0)) +
  scale_size_continuous(range = c(1,5)) +
```

```
  coord_fixed() +
  theme_graph() +
  theme(legend.position = "none")

verse <- which(V(vcomb_net)$name=="16:90")
v_name = V(vcomb_net)$name
fig913 = ggraph(vcomb_net, layout = "focus", focus = verse) +
  draw_circle(col = "darkblue", use = "focus",max.circle = 4) +
  geom_edge_link0(aes(edge_width = 1),edge_colour = "grey66") +
  geom_node_point(aes(fill="red", size=3), shape = 21) +
  geom_node_text(size = 3,label = v_name,
                 family = "serif", repel=T) +
  scale_edge_width_continuous(range = c(0.1,2.0)) +
  scale_size_continuous(range = c(1,5)) +
  coord_fixed() +
  theme_graph() +
  theme(legend.position = "none")

#### Word co-occurrence KG
library(text2vec)

## get the texts

surah87_o1 = kedges %>% filter(str_detect(from,"87:")) %>% select(from,to)
surah87_o2 = kedges %>% filter(from %in% surah87_o1$to) %>% select(from,to)
surah87_o3 = kedges %>% filter(from %in% surah87_o2$to) %>% select(from,to)
surah87_o4 = kedges %>% filter(from %in% surah87_o3$to) %>% select(from,to)
surah87_o5 = kedges %>% filter(from %in% surah87_o4$to) %>% select(from,to)
surah87_out = bind_rows(surah87_o1,surah87_o2,surah87_o3,surah87_o4,surah87_o5)

ala_intxt = quran_en_sahih %>% filter(ayah_title %in%
                          c(surah87n_l5$from,surah87n_l5$to)) %>% pull(text)
ala_outxt = quran_en_sahih %>% filter(ayah_title %in%
                                    c(surah87_out$from,surah87_out$to)) %>% pull(text)
## clean the texts
alain = tolower(ala_intxt)
alain = gsub("[^[:alnum:]\\-\\.\\s]", " ", alain)
alain = gsub("\\.", "", alain)
alain= trimws(alain)
## 1. tokenize
alain_ktoks = space_tokenizer(alain)
alain_itktoks = itoken(alain_ktoks, n_chunks = 10L)
stopw = stop_words$word
## 2. create vocabulary
alain_kvocab = create_vocabulary(alain_itktoks, stopwords = stopw)
## 3. vectorize the vocabulary
alain_k2vec = vocab_vectorizer(alain_kvocab)
## 4. create the tcm
alain_ktcm = create_tcm(alain_itktoks, alain_k2vec, skip_grams_window = 5L)
## 5. generate the Global Vector with rank = 50L
alain_kglove = GlobalVectors$new(rank = 50, x_max = 6)
## 6. Fit the GloVE model
alain_wv_main = alain_kglove$fit_transform(alain_ktcm, n_iter = 20)
## Use the model for prediction
alain_wv_context = alain_kglove$components
## This is in data.table format
alain_word_vec = alain_wv_main + t(alain_wv_context)

topic0 = alain_word_vec["allah",,drop = FALSE]
topic00 = alain_word_vec["lord",,drop = FALSE]
topic1 = alain_word_vec["living",,drop = FALSE]
topic2 = alain_word_vec["dying",,drop = FALSE]
topic3 = alain_word_vec["purify",,drop = FALSE]
topic4 = alain_word_vec["hell",,drop = FALSE]

topic5 = alain_word_vec["allah",,drop = FALSE] +
  alain_word_vec["lord",,drop = FALSE] +
  alain_word_vec["living",,drop = FALSE] +
  alain_word_vec["dying",,drop = FALSE] +
  alain_word_vec["purify",,drop = FALSE] +
  alain_word_vec["hell",,drop = FALSE]

t0_sim = sim2(x = alain_word_vec, y = topic0, method = "cosine")
htin0 = names(head(sort(t0_sim[,1], decreasing = T),7))
t00_sim = sim2(x = alain_word_vec, y = topic00, method = "cosine")
htin00 = names(head(sort(t00_sim[,1], decreasing = T),7))
t1_sim = sim2(x = alain_word_vec, y = topic1, method = "cosine")
htin1 = names(head(sort(t1_sim[,1], decreasing = T),7))
t2_sim = sim2(x = alain_word_vec, y = topic2, method = "cosine")
htin2 = names(head(sort(t2_sim[,1], decreasing = T),7))
```

```
t3_sim = sim2(x = alain_word_vec, y = topic3, method = "cosine")
htin3 = names(head(sort(t3_sim[,1], decreasing = T),7))
t4_sim = sim2(x = alain_word_vec, y = topic4, method = "cosine")
htin4 = names(head(sort(t4_sim[,1], decreasing = T),7))
t5_sim = sim2(x = alain_word_vec, y = topic5, method = "cosine")
htin5 = names(head(sort(t5_sim[,1], decreasing = T),7))

## clean the texts
alain = tolower(ala_outxt)
alain = gsub("[^[:alnum:]\\-\\.\\s]", " ", alain)
alain = gsub("\\.", "", alain)
alain= trimws(alain)
## 1. tokenize
alain_ktoks = space_tokenizer(alain)
alain_itktoks = itoken(alain_ktoks, n_chunks = 10L)
stopw = stop_words$word
## 2. create vocabulary
alain_kvocab = create_vocabulary(alain_itktoks, stopwords = stopw)
## 3. vectorize the vocabulary
alain_k2vec = vocab_vectorizer(alain_kvocab)
## 4. create the tcm
alain_ktcm = create_tcm(alain_itktoks, alain_k2vec, skip_grams_window = 5L)
## 5. generate the Global Vector with rank = 50L
alain_kglove = GlobalVectors$new(rank = 50, x_max = 6)
## 6. Fit the GloVE model
alain_wv_main = alain_kglove$fit_transform(alain_ktcm, n_iter = 20)
## Use the model for prediction
alain_wv_context = alain_kglove$components
## This is in data.table format
alain_word_vec = alain_wv_main + t(alain_wv_context)

topic0 = alain_word_vec["allah",,drop = FALSE]
topic00 = alain_word_vec["lord",,drop = FALSE]
topic1 = alain_word_vec["living",,drop = FALSE]
topic2 = alain_word_vec["dying",,drop = FALSE]
topic3 = alain_word_vec["purify",,drop = FALSE]
topic4 = alain_word_vec["hell",,drop = FALSE]

topic5 = alain_word_vec["allah",,drop = FALSE] +
  alain_word_vec["lord",,drop = FALSE] +
  alain_word_vec["living",,drop = FALSE] +
  alain_word_vec["dying",,drop = FALSE] +
  alain_word_vec["purify",,drop = FALSE] +
  alain_word_vec["hell",,drop = FALSE]

t0_sim = sim2(x = alain_word_vec, y = topic0, method = "cosine")
htout0 = names(head(sort(t0_sim[,1], decreasing = T),7))
t00_sim = sim2(x = alain_word_vec, y = topic00, method = "cosine")
htout00 = names(head(sort(t00_sim[,1], decreasing = T),7))
t1_sim = sim2(x = alain_word_vec, y = topic1, method = "cosine")
htout1 = names(head(sort(t1_sim[,1], decreasing = T),7))
t2_sim = sim2(x = alain_word_vec, y = topic2, method = "cosine")
htout2 = names(head(sort(t2_sim[,1], decreasing = T),7))
t3_sim = sim2(x = alain_word_vec, y = topic3, method = "cosine")
htout3 = names(head(sort(t3_sim[,1], decreasing = T),7))
t4_sim = sim2(x = alain_word_vec, y = topic4, method = "cosine")
htout4 = names(head(sort(t4_sim[,1], decreasing = T),7))
t5_sim = sim2(x = alain_word_vec, y = topic5, method = "cosine")
htout5 = names(head(sort(t5_sim[,1], decreasing = T),7))

## clean the texts
ala_ctext = c(ala_intxt,ala_outxt)
alain = tolower(ala_ctext)
alain = gsub("[^[:alnum:]\\-\\.\\s]", " ", alain)
alain = gsub("\\.", "", alain)
alain= trimws(alain)
## 1. tokenize
alain_ktoks = space_tokenizer(alain)
alain_itktoks = itoken(alain_ktoks, n_chunks = 10L)
stopw = stop_words$word
## 2. create vocabulary
alain_kvocab = create_vocabulary(alain_itktoks, stopwords = stopw)
## 3. vectorize the vocabulary
alain_k2vec = vocab_vectorizer(alain_kvocab)
## 4. create the tcm
alain_ktcm = create_tcm(alain_itktoks, alain_k2vec, skip_grams_window = 5L)
## 5. generate the Global Vector with rank = 50L
alain_kglove = GlobalVectors$new(rank = 50, x_max = 6)
## 6. Fit the GloVE model
alain_wv_main = alain_kglove$fit_transform(alain_ktcm, n_iter = 20)
```

```
## Use the model for prediction
alain_wv_context = alain_kglove$components
## This is in data.table format
alain_word_vec = alain_wv_main + t(alain_wv_context)

topic0 = alain_word_vec["allah",,drop = FALSE]
topic00 = alain_word_vec["lord",,drop = FALSE]
topic1 = alain_word_vec["living",,drop = FALSE]
topic2 = alain_word_vec["dying",,drop = FALSE]
topic3 = alain_word_vec["purify",,drop = FALSE]
topic4 = alain_word_vec["hell",,drop = FALSE]

topic5 = alain_word_vec["allah",,drop = FALSE] +
  alain_word_vec["lord",,drop = FALSE] +
  alain_word_vec["living",,drop = FALSE] +
  alain_word_vec["dying",,drop = FALSE] +
  alain_word_vec["purify",,drop = FALSE] +
  alain_word_vec["hell",,drop = FALSE]

t0_sim = sim2(x = alain_word_vec, y = topic0, method = "cosine")
htino0 = names(head(sort(t0_sim[,1], decreasing = T),7))
t00_sim = sim2(x = alain_word_vec, y = topic00, method = "cosine")
htino00 = names(head(sort(t00_sim[,1], decreasing = T),7))
t1_sim = sim2(x = alain_word_vec, y = topic1, method = "cosine")
htino1 = names(head(sort(t1_sim[,1], decreasing = T),7))
t2_sim = sim2(x = alain_word_vec, y = topic2, method = "cosine")
htino2 = names(head(sort(t2_sim[,1], decreasing = T),7))
t3_sim = sim2(x = alain_word_vec, y = topic3, method = "cosine")
htino3 = names(head(sort(t3_sim[,1], decreasing = T),7))
t4_sim = sim2(x = alain_word_vec, y = topic4, method = "cosine")
htino4 = names(head(sort(t4_sim[,1], decreasing = T),7))
t5_sim = sim2(x = alain_word_vec, y = topic5, method = "cosine")
htino5 = names(head(sort(t5_sim[,1], decreasing = T),7))
```

10 Way Forward

There are voluminous forms of knowledge that had been passed down by the Islamic scholars, under the banner of Islamic sciences. Many of this knowledge had been cast as fixed and no new knowledge is possible; such is the case of closing the door of Ijtihad (independent reasoning)[133]. Similarly, we may ask, is there any new knowledge in the sciences of exegesis of Al-Quran, or the sciences of Al-Hadith?

Here we would like to distinguish between "new knowledge" and "new tools" for acquiring knowledge. The advancement of artificial intelligence (AI) has certainly received much attention of late. AI plays a major role in transforming Big Data into information, and from information to knowledge. Knowledge is where intelligence lies. We agree that machines can have intelligence. However, for machines to improve beyond intelligence towards wisdom, would be an impossibility. AI is an improvement in tools, through which we can improve our knowledge acquisition. But in the process, knowledge is for the benefit of humans, rather than machines.

Tools provide us baseline knowledge, that is sometimes used to confirm what we already know. For example, it is widely accepted that Allah is the central theme or subject matter of Al-Quran. The tools we used for Quran Analytics give clearer proof of that and, interestingly, a new perspective in the form of some unique characteristics of word networks. In this example, we say that the tools are providing affirmation to our knowledge and some new perspectives; but it does not add new knowledge. But tools can further be used beyond just existing known knowledge, such as to expand the analysis beyond its baseline. For example, we can further study why the name "Allah" is used in certain ways in the Al-Quran, through a bi-gram analysis. New insights from the analysis may yield new knowledge, which has not been discovered before. This is how tools are useful for exploring new knowledge.

Furthermore, we know that knowledge is dynamic. An example is the progress of knowledge through time. More discoveries are made as time passed. All that we want is to make sure that all knowledge is updated. AI presents the best modern tools to analyze data, because machines can scan, study, and look upon any possible large source of data, information, and knowledge, with fewer errors or fatigue (as we humans do).

[133] http://www.oxfordislamicstudies.com/article/opr/t125/e990

As an example, it is possible with AI tools today to study the differences between Ibnu Sina and Ibnu Rushd, who lived geographically far apart and centuries apart in time. Furthermore, the differences between them can be studied within the context of current-day knowledge and scholars.

In general, we say that AI tools help us to reaffirm and refine the "old existing knowledge" of Islam that we know, whether they are in the forms of traditions, fatwas, and interpretations. Furthermore, new tools can further strengthen the knowledge base. This is achieved by deployment of the new tools on existing old knowledge and re-represent them using the new tools. Examples of this include new visualization techniques, presentation in statistical forms or analysis, and summarization of knowledge in an abstract manner. Certainly, using new tools like text analytics on existing "data" like the Quran should reveal new knowledge and understanding about the Quran. It is also proof of the eternal relevance of the Quran as a book of knowledge.

New tools for studying Al-Quran

Studying Al-Quran is an important subject for Muslims and non-Muslims alike. In this book, we have been exploring the idea of mooting a specialized area called "Quran Analytics". In various chapters, we have repeated the exercise of using NLP tools for different applications on the English translations of Al-Quran, in particular Saheeh International and Yusuf Ali. Throughout the various chapters, we keep highlighting various affirmations of knowledge of Al-Quran, which is already accepted as common knowledge. The concept of "Basheeran" (glad tidings) and "Nazeeran" (warnings) for the case of sentiment analysis is an example.

The other emphasis that we have been making is to highlight the approach of using the tools in search of discoveries. For this purpose, we introduced tools that are commonly used in NLP exercises, such as word collocations, word networks, and word modelings. Since the focus is on the translated Al-Quran, the analysis used is tied to the English structure of the translations. Despite these limitations, many interesting insights were obtained. As an example, we could detect the resiliency of the messages in Al-Quran, regardless of the translation method used. Despite differences in time, Yusuf Ali and Saheeh contain similarities in many structures that are attributed mainly to the original source of the texts, which is the Arabic Al-Quran.

What we have attempted represents only some sampling of analyses with the application of new tools in NLP applied on the English translations of Al-Quran. We have not even attempted to deal with the Quranic Arabic language, which is another vast area by itself. It is known that the Quranic Arabic language influences the Classical Arabic language which in turn was the base for most of the works in Islamic knowledge. The language is not the creation of Muslims, but rather was passed down from the Arabs, making it the lingua-franca of Islam and Muslims. The Quranic language dominated the scientific and knowledge dissemination of the world for over a thousand years. All knowledge and scientific works, except for literary pieces (particularly among the Persians) were in the Arabic language. The sciences of the Arabic language (a major issue in Computational Linguistics and Natural Language Processing today), evolved into well-established sciences very early on in the history of the language. This is evident from the early works of Al-Farahidi, followed by his followers, among them is Al-Sibawayhi. What they had accomplished is to establish the rules of grammar for the Arabic language.

Among the task of CL and NLP nowadays is to convert the rules of language into a computational model. There no better sources that we can rely upon to develop the models (for Classical Arabic) than these classical works. By right, these works (and works of many scholars in later periods), present a gold mine of data and process, which can be converted into concise computational models of the language.

A full-scale work involving the Quranic Arabic language, the various sources for the knowledge on Al-Quran (or Ulum Al-Quran), deploying various tools of CL and NLP, and development of models and applications for knowledge of Al-Quran, is what we termed as "Quran Analytics". This is what we envisaged to be developed in its full capacity.

Limitations

Throughout the book we have demonstrated many applications of CL and NLP on Al-Quran, using mainly the selected English translations of Saheeh and Yusuf Ali. Admittedly, the selection and focus are narrow and limited. The idea is to demonstrate some of the tools using the **R** programming language. Despite these limitations, we need to emphasize that what we have demonstrated is the ability of the tools to expand the knowledge further.

Our focus in this book is mainly on applying the tools and modeling, but not on the meaning and interpretations. Neither do we have a focus on the aspect of the language of Al-Quran. However, given a domain knowledge on a particular area, like a method of exegesis of Al-Quran such as by Imam Ibnu Katheer, we can quickly expand the subject with ease as shown in Chapter 9.

The ego network of verses from Tafseer Ibnu Katheer reveals many of his understanding of Al-Quran which is not as evident if we read the works verbatim. The only way we can see this is through the network representations. Even then, our work so far is to record and display, without proposing deeper meanings and interpretations of the arrangements. We purposely wanted to avoid the interpretations since it requires deeper knowledge of the subject under discussion.

In any case, we would say that the limitations of our works are only by choice rather than by default. Any interested student or researcher can take the direction which we started with and make many other findings and learnings from the old knowledge as well as discoveries of new knowledge.

Direction of future works

It is our belief, based on the findings in this book, that the direction of future research for Quran Analytics is tremendous and vast. This is encouraged by two facts. First, the CL and NLP tools are continually improving which will make many more tools and methods available for applications on Quran Analytics. Second, the amount of digitized data becoming available is ever increasing which allows us to perform more analysis based on classical sources of knowledge as well as contemporary sources. All are now becoming more easily available.

The works however require caution, since implementations of modern tools are not value-free. For example, sentiment analysis based on English texts assume certain value judgments which may not fit the Quranic language. Therefore, there is also a real need of developing models from scratch, instead of relying on ready-made tools.

The research approach has been towards "open and reproducible" research. What is meant is basically that research is encouraged to use openly sourced data, openly sourced codes (or programming languages), and should be available for inspection and reproduction by others. We fully agree and adhere to this approach, whereby all the codes of our works are openly available on Github, written mainly using **R** programming language as an open-source tool.

We also follow closely the works of development of corpora for the Islamic texts and encourage development along those lines. We also would like to invite more people, especially those with domain knowledge in Islamic sciences to adopt a similar approach of using AI tools for Islamic sciences applications.

Concluding remarks

Our sincere hope is to spur more works and usage of AI tools for the furtherance of the development of Islamic knowledge. Quran Analytics, in our view, is the starting point, and this book is an introduction to the subject. There are many lessons and improvements possible using modern AI tools for Islamic knowledge.

For this field to expand, we believe that it is important for people from various disciplines and backgrounds to collaborate. It requires people with a background in computing and AI, with people having knowledge

of Islamic sciences, and knowledge of Classical Arabic language, combined. This inter-disciplinary effort is a must if progress and improvements are desired.

We call upon everyone to join us in these efforts, and we pray that Allah will reward all of us in this endeavor.

References

References

Abdul-Baqi, M. F. (1945). *Al-Mu'jam Al-Mufahras Li Alfaz Al-Quran Al-Kareem*. Dar Ahi'a Al-Tirath Al-Arabi, Beirut, Lebanon.

Al-Saffar, A., Awang, S., Tao, H., Omar, N., Al-Saiagh, W., and Al-bared, M. (2018). Malay sentiment analysis based on combined classification approaches and senti-lexicon algorithm. *PLOS ONE*, 13(4):1–18.

Alsuwaidan, T. and Hussin, A. (2021). Islam simplified: A holistic view of the quran. Unpublished Manuscript.

Arnold, T. B. (2016). *cleanNLP: A Tidy Data Model for Natural Language Processing*.

Arnold, T. B. and Tilton, L. (2016). *coreNLP: Wrappers Around Stanford Corenlp Tools*.

Atwell, E. (2018). *Classical and Modern Arabic Corpora: genre and language change*. John Benjamins Publishing Company, Amsterdam, The Netherlands. ISBN 9789027201485.

Baayen, H. R. (2008). *Analyzing Linguistic Data: A practical introduction to statistics using R*. Cambridge University Press, Cambridge, United Kingdom.

Ban, K., Meštrović, A., and Martincic-Ipsic, S. (2014). Initial comparison of linguistic networks measures for parallel texts.

Bannister, A. G. (2014). *An Oral Formulaic Study of the Quran*. Lexington Books, Plymouth, United Kingdom.

Barabasi, A.-L. (2016). *Network Science*. Cambridge University Press, Cambridge, Massachussets. ISBN 978-1-107-07626.

Benoit, K., Watanabe, K., Wang, H., Müller, S., Perry, P. O., Lauderdale, B., Gruber, J., Lowe, W., and Sindhwani, V. (2020). *quanteda.textmodels: Scaling models and classifiers for textual data*.

Benoit, K., Watanabe, K., Wang, H., Nulty, P., Obeng, A., Müller, S., and Matsuo, A. (2018). *quanteda: An R package for the quantitative analysis of textual data*.

Blaheta, D. and Johnson, M. (2001). From taxonomies over ontologies to knowledge graphs. Available at http://web.science.mq.edu.au/~mjohnson/papers/2001/dpb-colloc01.pdf (2021/02/16).

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research*, 21(3):993–1022.

Blumauer, A. (2016). From taxonomies over ontologies to knowledge graphs. Available at https://semantic-web.com/2014/07/15/from-taxonomies-over-ontologies-to-knowledge-graphs/ (2021/02/16).

Chambers, J. M. (2016). *Extending R*. Chapman Hall/CRC, Boca Raton, Florida.

Csárdi, G. (2020). *igraph: Network Analysis and Visualization*.

Dawson, C. W., Ghallab, A., Mohsen, A., and Ali, Y. (2020). Arabic sentiment analysis: A systematic literature review. *Applied Computational Intelligence and Soft Computing*, 2020:7403128.

DiMarco, C. and Hirst, G. (1988). Stylistic grammars in language translation. *COLING*.

Dukes, K. and Habash, N. (2010). Morphological annotation of quranic arabic. In *Proceedings of the 7th International Conference on Language Resources and Evaluation, LREC 2010*, pages 2530–2536. European Language Resources Association (ELRA).

Feinerer, I., Hornik, K., and Software, A. (2020). *tm: Text Mining Package*.

Gries, S. T. (2017). *Quantitative Corpus Linguistics with R: A Practical Introduction*. Routledge, Taylor and Francis Group, New York, New York, 2nd edition.

Grimmer, J. and Stewart, B. M. (2013). Text as data: The promise and pitfalls of automatic content analysis methods for political texts. *Political Analysis*, 21(3):267–297.

Grün, B., Hornik, K., Blei, D. M., Lafferty, J. D., Phan, X.-H., Matsumoto, M., Nishimura, T., and Cokus, S. (2020). *Topic models*.

Heiss, A. (2018). *quRan: Complete Text of the Qur'an*.

Hirschberg, J. and Manning, C. D. (2015). Advances in natural language processing. *Science Magazine*, 349:261–266.

Izutsu, T. (1964). *God and Man in the Qur'an*. Keio University, Tokyo, Japan.

Jackson, M. O. (2008). *Social and Economic Networks*. Princeton University Press, Princeton,New Jersey, USA.

Jelodar, H., Wang, Y., Yuan, C., Feng, X., Jiang, X., Li, Y., and Zhao, L. (2019). Latent dirichlet allocation (lda) and topic modeling: models, applications, a survey. *Multimedia Tools and Applications*, 78(11):15169–15211.

Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, New Jersey, United States, 2nd edition. ISBN 9780130950697.

Kolaczyk, E. and Csardi, G. (2014). *Statistical Analysis of Network Data with R*. Springer, New York,New York, USA.

Landauer, T. K., Foltz, P. W., and Laham, D. (1998). An introduction to latent semantic analysis. *Discourse Processes*, 25:259–284.

Manning, C. D., Raghavan, P., and Schutze, H. (2009). *An Introduction to Information Retrieval*. Cambridge University Press, Cambridge, England.

Manning, C. D. and Schutze, H. (1999). *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts.

Marie-Sainte, S. L., Alalyani, N., Alotaibi, S., Ghouzali, S., and Abunadi, I. (2019). Arabic natural language processing and machine learning-based systems. *IEEE*, 7:7011–7019.

Paulheim, H. (2016). Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web Journal*, Preprint:1–20.

Pedersen, T. L. (2017). *ggraph: An Implementation of Grammar of Graphics for Graphs and Networks*.

Peng, R. D. (2014). *R Programming for Data Science*. Leanpub, Victoria, British Columbia, Canada.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Popper, K. (1992). *The Logic of Scientific Discovery*. Routledge Classics, London, United Kingdom. ISBN 0-203-99462-0.

Queiroz, G. D., Fay, C., Hvitfeldt, E., Keyes, O., Mastny, T., Erickson, J., Robinson, D., Misra, K., and Silge, J. (2020). *tidytext: Text Mining using 'dplyr', 'ggplot2', and Other Tidy Tools*.

Rahman, F. (1980). *Major Themes of the Qur'an*. The University of Chicago Press, Chicago, Illinois, USA.

Reinanda, R., Meij, E., and de Rijke, M. (2020). Knowledge graphs: An information retrieval perspective. *Foundations and Trends in Information Retrieval*, 14(4):289–444.

Rinker, T. W. (2017). *sentimentr: Calculate Text Polarity Sentiment*.

Roberts, M., Stewart, B., and Tingley, D. (2019). stm : An r package for structural topic models. *Journal of Statistical Software*, 91.

Roberts, M., Stewart, B., Tingley, D., and Benoit, K. (2020). *stm: Estimation of the structural topic model*.

Roberts, M. E., Stewart, B. M., Tingley, D., and Airoldi, E. M. (2014). The structural topic model and applied social science. *American Journal of Political Science*, 58:1064–1082.

Robinson, D., Misra, K., and Silge, J. (2020). *widyr: Widen, Process, then Re-Tidy Data*.

Saeh, B. (2015). *The Miraculous Language of Qur'an: Evidence of Divine Origin*. International Institute of Islamic Thought, London, United Kingdom. ISBN 978-1-56564-665-0.

Saheeh-International (1997). *The Quran: Arabic Text with Corresponding English Meanings*. Abul Qasim Publishing House, Jeddah, Saudi Arabia. ISBN-13: 978-9960792637.

Selivanov, D., Bickel, M., and Wang, Q. (2020). *text2vec: Modern text mining framework for R*.

Silge, J. and Robinson, D. (2017). *Text Mining with R: A Tidy Approach*. O'Reilly Media, Inc., Newton, Massachussets. ISBN: 9781491981658.

Singhal, A. (2012). Introducing the knowledge graph: things, not strings. Available at https://blog.google/products/search/introducing-knowledge-graph-things-not/ (2021/02/16).

Von-Denfer, A. (1983). *Ulum Al-Quran: An Introduction to the Sciences of the Quran*. The Islamic Foundation, London, United Kingdom.

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., Woo, K., Yutani, H., Dunnington, D., and RStudio (2020). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*.

Wickham, H. and Grolemund, G. (2016). *R for Data Science*. O'Reilly Media, Inc, Newton, Massachussets.

Wijffels, J. and BNOSAC (2020). *textrank: Summarize Text by Ranking Sentences and Finding Keywords*.

Wijffels, J., Straka, M., and Straková, J. (2020). *udpipe: Tokenization, Parts of Speech Tagging, Lemmatization and Dependency Parsing with the 'UDPipe' 'NLP' Toolkit*.

Yusuf-Ali, A. (2003). *The Meaning of the Holy Quran*. Amana Publications, Beltsville, Maryland. ISBN 9781590080160.

Zipf, G. K. (1949). *Human Behavior and the Principle of Least Effort*. Addison-Wesley, New York, New York.