INTERNATIONAL
HELLENIC
UNIVERSITY

Plagiarism Detection in Text Collections

Panagiotis Kalampokis

SID: 3306150007

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Mobile and Web Computing

November 2016

THESSALONIKI – GREECE

Plagiarism Detection in Text Collections

Panagiotis Kalampokis

SID: 3306150007

Supervisor:	Prof. Apostolos Papadopoulos
Supervising Committee Members:	Assoc. Prof. Name Surname
	Assist. Prof. Name Surname

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Mobile and Web Computing

November 2016

THESSALONIKI – GREECE

To my beloved Father who gives his example.

To my beloved Mother who gives her advice.

To the Runner who keeps failing…

Abstract

This is a dissertation thesis submitted for the degree of MSc in Mobile and Web Computing at the International Hellenic University.

The main purpose of this dissertation was to find an efficient way to compare a big corpus of document texts among them and check which of them have been subjected plagiarism.

We will walk through a set of algorithms which are used for calculating the similarity between a set of documents, and we will conclude to an algorithm that is used most, for big data sets, (the MinHash algorithm).

The MinHash algorithm makes extensive use of Hashing functions so as to reduce the dimensionality space kept for the “useful” part of a document during the action of preprocessing, and estimates the probability, that two documents resemble each other with the LSH technique. We will discuss what “useful” part means along the way while explaining the algorithm.

First of all I would like to express my gratitude to my supervisor and professor dr. Apostolos Papadopoulos, for the support and guidance he offered along the way, to the successful completion of my dissertation.

Panagiotis Kalampokis

11/2016

Contents

1 Introduction

Plagiarism is the "wrongful appropriation", "stealing and publication" of another author's "language, thoughts, ideas, or expressions" and the representation of them as one's own original work. [2]

1.1 Plagiarism Definition

Plagiarism etymology comes from the very beginning of the 1st century, from the use of the Latin word *plagiarius* (literally *kidnapper*) to denote stealing someone else's work. [3] Plagiarism definition does have many uses and many references from every part of the earth. Plagiarism in some extreme cases takes the form of copyright infringement but it is not considered a crime. Plagiarism is considered an ethical offense in academia and industry, and calls for various sanctions like penalties, suspension or even expulsion in some extreme cases.

In this thesis we will try to address the plagiarism plague that has made its presence more and more frequent nowadays, thereby deteriorating the situation of copyright- infringement. With the widespread presence of Plagiarism, a degeneration of intellectual property and creation has become more apparent.

1.2 Plagiarism detection

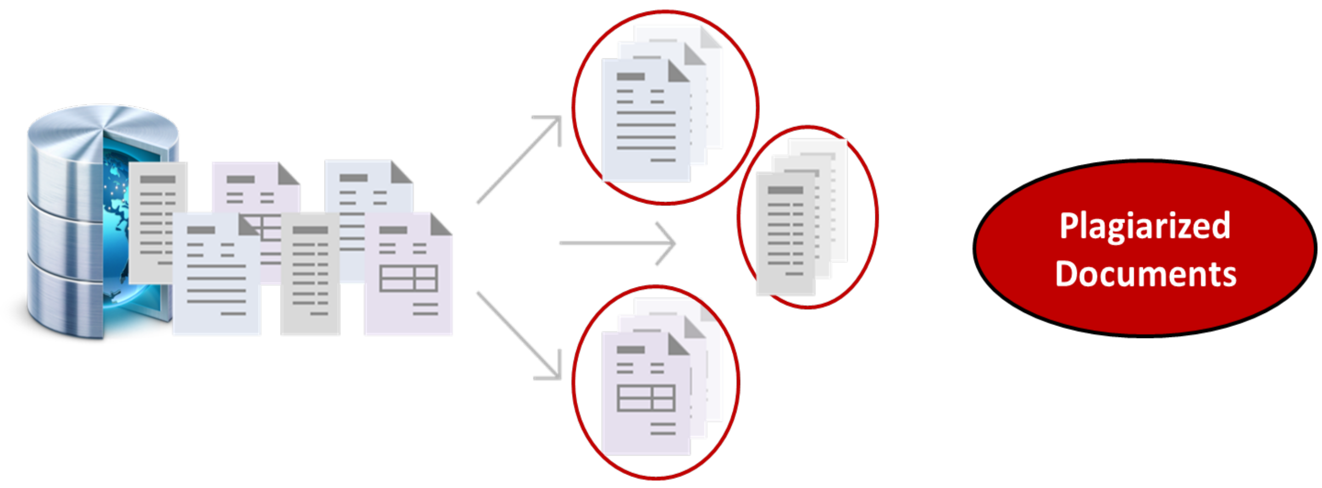
Plagiarism has made its presence almost everywhere nowadays and has significantly evolved in the past few years. Academia, scientific papers, documents, essays, reports and source code are some of the examples where plagiarism is found in abundance.

Easy access to vast information and the Internet have remarkably helped plagiarism to rise to a totally higher level than before. The need of plagiarism detection software has also risen significantly with the outbreak of intellectual property theft and the more effortless ways to do it.

With the huge evolution of technology more and more plagiarism detection software programs are making their appearance. They are very practical and way more efficient in detecting plagiarized references or even source code by comparing millions or even billions of files in a very short time frame, than the inefficient old manual detection way. There is a lot of software – computer assisted plagiarism detection software which is known as CaPD(computer assisted plagiarism detection software). This Computer assisted plagiarism detection software, belongs to Information Retrieval approach of finding similar text documents.

1.3 What is our Problem?

Figure 1.1: Depiction of our problem.



Our problem breaks down as follows:

From a huge corpus of Documents (e.g 10 million), find all the plagiarized documents with above a percentage: x% of similarity.

- Artlessly, we would need to calculate all the Jaccard similarities between pairs of documents for every document dual set. Jaccard similarity coefficient, measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets: $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$. [7]

- That is $\frac{N \times (N-1)}{2} \approx 5 \, * \, 10^{11} \, contrasts$! That is a very high number of comparisons. For a generally common hardware pc, it would take more that ¾ of the week to compute all the pairwise Jaccard similarities between the documents!
- Imagine how much time it would take for a common hardware to calculate all the pairwise Jaccard similarities between the documents if we had 11 million documents. it would take more than 365 days!
- We will see how this can be done in O(n) in following chapters.

Now let’s proceed with the definition of similarity.

2 Similarity

Usually when we talk about similarity, we are trying to measure how similar two Sets are. What we mean by that is that if we represent these sets with a metric (numeric preferably), we can then measure the distance between them using one of the distance metrics that exist. There are many definitions of similarity and many similarity and distance metrics exist. Below we give some of them:

- Euclidean distance : d(p, q) = d(q, p) = $\sqrt{\sum_{i=1}^{N}(qi \, - \, pi)^2}$.
- Cosine distance and Cosine Similarity.
- Hamming distance.
- Jaccard Similarity, etc.

It is useful to note that Cosine Similarity is used commonly in information retrieval systems and in data mining problems. We will take a closer look though in Jaccard similarity as it is more appropriate in our case. Let’s begin with the similarity of sets now and we will expand later in more detail.

2.1 Similarity of Sets

One of the most important data-mining problems is to find similar sets from a huge collection of items.

For example we could compare a big corpus of texts for plagiarism, that is finding quite similar texts that have been subjected to plagiarism. Or another example would be to compare a huge corpus of images to find near duplicate images.

Generally speaking there are too many examples in real life that we can use data mining techniques to tackle a big number of problems.

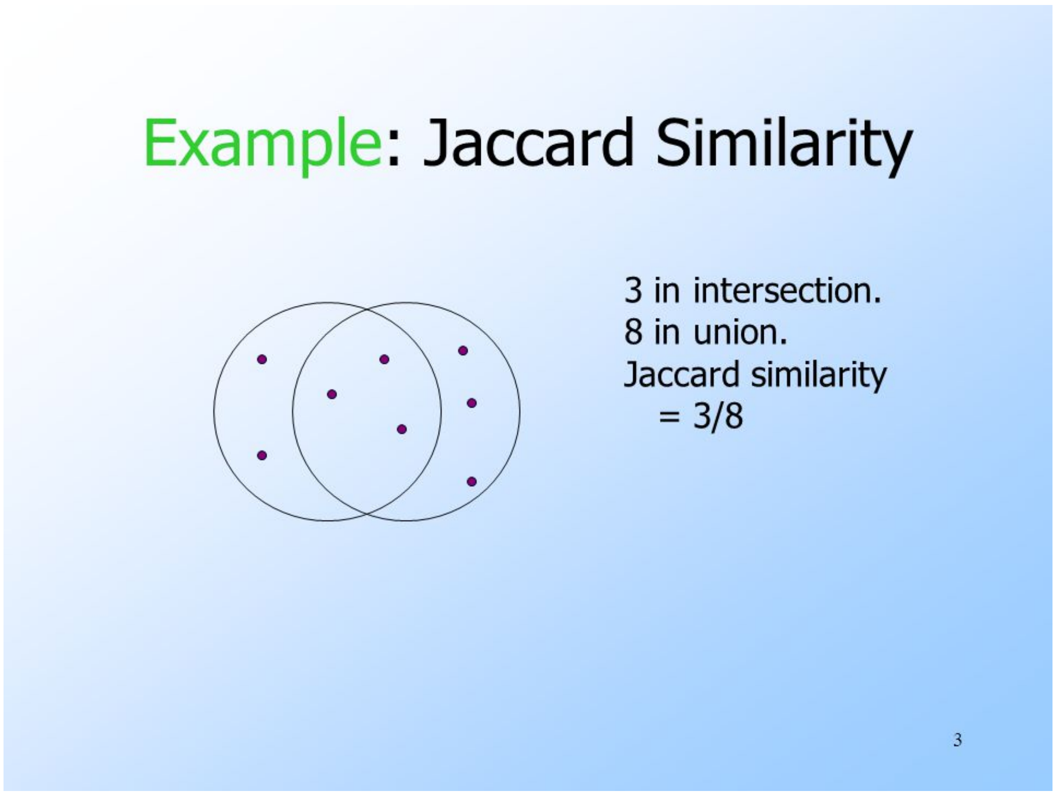
What we are after with “Similarity” meaning is: from two arbitrarily sets from our large corpus of documents, find those sets, which have as many common elements as they could. We then use the Shingling method to extract shingles from the initial sets. A Shingle is a k-gram character set. We will talk more about it in the following chapters. Then we pass all the k-grams from the Minhashing method, that is extracting the Min-Hash Signature as it is called from the original document. This is a highly efficient and applicable method for significantly reducing the size of the original document, while preserving in high degree the similarity between all the documents in the corpus. We will talk about this more extensively in the following chapters.

Finally we will talk about the Locality Sensitive Hashing technique, which is the last and most significant piece which compounds the whole artifact. That is how the LSH technique finds the most similar pairs in one pass instead of searching all the pairs for similarity. We will explain that also in more detail in the next chapters.

2.2 Jaccard Similarity

Let’s start with a definition of the Jaccard Similarity and we will continue with the big picture of the MinHashing technique. Jaccard Similarity between two sets is defined as the Intersection divided by the Union of the elements of two sets. That is the number of common elements divided by the number of all the elements in the two sets. Below is a picture that represents it quite accurately.

Figure 2.1: Jaccard similarity between two Sets.



Example: Jaccard Similarity

3 in intersection.
8 in union.
Jaccard similarity
= 3/8

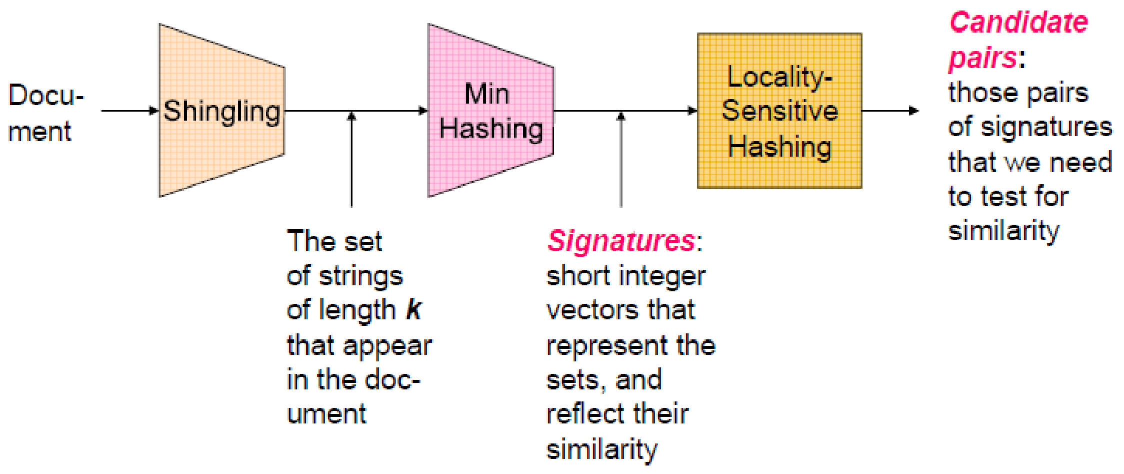
3

From the above example we can easily deduce that there are two Sets. Let’s say A is the left one and B the right one that have three elements that reside in the same space which belongs to both sets A and B, that is the intersection sets. |Intersection(A,B)| = 3. We can also note that there are two elements in A which belong exclusively in A set and not in B. Also there are three elements that exist exclusively in B and not in A. So the union of A and B is : |Union(A,B)| = 2 + 3 + 3 = 8. That is the all the elements in A and B. So the Jaccard Similarity between those sets is: Jaccard Similarity(A,B) = $\frac{3}{8}$.

In order to find similar text sets between documents, we need to somehow quantify these text sets and measure their similarity. We accomplish this with the “shingling” technique.

But before we get into the inner workings of any approach or technique, let’s take a look at the following diagram which gives us, a big picture of what we are about to see and analyze in the following chapters.

Figure 2.2 : The Big picture of MinHash – LSH



Docu-ment
Shingling
Min Hashing
Locality-Sensitive Hashing
Candidate pairs: those pairs of signatures that we need to test for similarity
The set of strings of length k that appear in the document
Signatures: short integer vectors that represent the sets, and reflect their similarity

What the above FIGURE 3.1 shows us, is that every document passes through these 3 phases:

1. Shingling phase: We extract the k-grams from the documents.
2. MinHashing phase: We build the minhash signature from the sets while maintaining the resemblance of the sets.
3. Locality-Sensitive Hashing: We extract candidate pairs from the previous Signature Matrix.

We will have this big picture in mind in all following chapters, as we explain every chapter in more detail soon. The MinHash algorithm that we are about to scrutinize very closely and implement afterwards in the source code, is based on the book: Mining of Massive Datasets [1]. Now let’s continue with the Shingling phase.

3 Shingles

First of all, we will try to give an explanation of why we convert big documents to k-gram sets, that is k-character length sets. The extraction of k-shingles from the documents is a very good approach for finding the similar sets afterwards, regardless of whether we are addressing plagiarism problem or near duplicate document problem inside our corpus, because even if we mix words or sentences inside a document, it will still find a big intersection set between the plagiarized documents. That is the Shingling technique.

Every document is a string set of characters. K-Shingling is a sliding string window of k character length in the document that covers the whole document, thereby creating a set of k-shingles for each document.

3.1 Shingling size

The selection of k in k-character shingles is very important. So what we mean by k-character shingle: for example if k = 2 => {he},{el},{ll}, {lo},… then we would have bigrams.

You can quickly see that if k is too small we will have a big number of intersected elements between two sets, but that number will drive us to false results. In a nutshell, all documents will be found plagiarized, or if we had Web pages then they would have high Jaccard Similarity because all the characters that appear in a Web page are similar to the most common characters a web page may have, which is not a good result.

How large k should be?

That is a good question that arises when we deal with text sets. First of all, we have to ask ourselves how big are the text sets? Are we dealing with big documents that are about 10 pages or more long? Are we dealing with scientific papers or theses? Or are we comparing a big corpus of e-mails from a server or possibly more servers?
Generally speaking, what is important to us for two sets (for example, a bag of words), is that if we take a random k-character shingle from one set, the possibility for this same k-character shingle to belong to the other set too, should be very small, unless they are “plagiarised” or have many elements in common. For small bag of words like e-mails k

= 5 is considered a good choice, whether for bigger documents like in our case, a good choice is to take k = 9 or k = 10.
That has to do with the approximate calculation of how many different n-grams you expect to see from the bag of words you are checking. For example how many different k-grams do we expect in a common e-mail? Probabilistically, it is well known that a common e-mail is much smaller than 14 million characters long. That means if we have 27 characters in English Alphabet, then: 27^5 = 14,348,907. That means that if we pick k = 5 for emails, it would be a very good and accurate choice for comparing two e-mails. Accordingly if our corpus is big texts, like documents of 10 pages or more long, k = 9 or k = 10 is proven to be a very good approach to our problem. In our case that we pick k = 9 character long shingles, it is deemed a proper value for the solution for our problem. Let’s move on now, to how we manipulate the shingles.

3.2 Hashing shingles

Since all the shingles will pass through computer manipulation, it is better to represent all the shingles, as numbers. Numbers can be manipulated, and calculated much better than raw k-character shingles. In order to do that, we need a good hash function that takes raw k-character shingles, and returns a highly representative integer. In our case (and in the source code) we use the MurMurHash 3 Hash function which gives very good results in hashing strings.

This hash function takes a string as an argument, and returns a number which is from (0 – a number of buckets we have chosen) which represents uniquely a shingle of length k. So we expect from this “good” hash function that two different shingles of length k, to return two different distinct numbers, and return the same number if and only if these two k-string length shingles are the same. That is why we need a good hash function which maps k-character shingles to a big number of buckets, which are essentially numbers in the range we have pre-configured. In our case (0 – number of total buckets which is $2^{31} - 1$). Here we have strongly connected the term shingle with the term bucket, and I will explain what we mean with that.

What we mean here is that after the Hashed version of a k-character shingle to a specific number, this specific number, now represents a specific bucket number in our space. This space is defined to contain all the possible k-character shingles that we can possibly make. This is a finite space of all the possible k-gram shingles. So if we represent this space a.k.a: Universe of k-gram shingles with numbers, we have a finite Universe of numbers. That is the bucket numbers. Every distinct k-shingle represents a bucket number.

With this representation of k-character shingles to integers, we can now use only 4-bytes (the common length for every integer represented by a machine) for every k-gram shingle. In this way we can manipulate k-character shingles much more efficiently with numbers than with characters. Computer calculations and manipulations to plain numbers are much more efficient than String manipulations, and 4-byte standard size for every number gives better computing quality in computing actions.

3.3 Preserve similarity of sets while shrinking the space

Each shingle, which is a 9-character length string, is now represented by a number which holds the “distinct” property. Even if every number takes no more than 4-bytes and this is a good plus for operational, computational and storage efficiency in computing, if we split the document into a set of shingles (9 character string length in our case), the number of shingles remains prohibitively high. For example if we take a document containing of 10.000 characters, and we split it to 9 shingle character string length, we get a set of 9,992 shingles! This set is still very large! Even using 4-bytes for each k-gram shingle, the number of shingles saved for every document remains prohibitively high. That means the space for storage remains very large. Therefore imagine how much data storage we would have to retain if we had millions of documents. That would be extremely high. Moreover, we haven’t achieved a big step so far, because we would still need to calculate all the pairwise Jaccard Similarities between all those sets.

Our goal is to reduce the amount of these large sets, to smaller sets that represent them, with as high as possible “resemblance” rate, we can achieve. We call this reduced amount of every set, a “signature”.

The most significant property for each representative signature extracted from every single document that has to hold, is that the true similarity percentage between two documents has to be about equal to the similarity percentage of their corresponding signatures. And what we mean by Similarity percentage here, is the Jaccard similarity. So we can arrive at the following property: JS(Doc1 , Doc2) ≈ JS(Sig(Doc1),Sig(Doc2)). Where JS = Jaccard Similarity and Sig = Signature. It turns out to hold that the larger the signatures we keep, that is the more elements we keep for the signatures, the more accurate the similarity percentage we will find for the examined sets. So we can write that:

$$\text{JS}(\lim_{\#elem \to N}(Sig_1\,,\ Sig_2\,) \approx \text{JS}(Sig_1\,,\ Sig_2\,)\ , \qquad \text{N} >>$$

That is for a natural big number N, the approximate Jaccard similarity of two signatures of two sets, is about equal the true Jaccard Similarity of those sets. We will talk about how big N should be, and how we pick it, to give us a good approximation in later chapters when we talk about the LSH technique.

One naïve approach to reduce a set of shingles and make a signature, is to pick (say) 200 randomly selected shingles from one document, and compare with another 200 randomly selected shingles from another document with Jaccard Similarity.

We will see that this is very close to what we are doing with Minhash Signature Matrix that shortly follows, but with a different selection technique of Shingles.

Let's take for example a document that is about 30.000 characters long. It is easy to see that from this document we can extract 30.000 – k + 1 shingles. So if we pick k = 9, we extract 30.000 – 9 + 1 = 29.992 9-gram shingles. Then we boil that number down to 200 randomly selected shingles from our initial set of 29.992. You can imagine that, by randomly selecting 200 shingles from our initial set, we can represent our newly much shorter of 200 shingles set as our new Signature for this set. We will shortly see that the random selection of 200 shingles is not actually done by this way (randomly), but you can temporarily assume this for now. Then we can suppose we have a "quite representative" signature from our initial set. And you can suppose also that the bigger the signature, that is the more elements you take as your signature for representation of a set, the more accurate this would be.

So what we have done so far is reducing significantly the initial big set of 29.992 shingles in our example, which are not actually shingles but plain numbers after they pass from the initial Hash- Function to 200 integer signature.

Imagine now, that we have another document of 50.000 characters long. Following the same procedure as above from 50.000 – 9 + 1 = 49.992 shingles, we also take 200 random integers to construct our 200 integer signature from the second document. Now instead of comparing 29.992 shingles from the first document with 49.992, the second document, we are comparing a set of 200 integers with another set of 200 integers. In fact, we are comparing the signatures themselves instead of the original documents.

What we have achieved so far is that we managed to significantly reduce the workload, the space needed for storage and we improved the efficiency of the computations in great extent by using integers instead of strings. Please note that the signature of a document doesn't represent the initial document accurately 100%. We will talk about it in more details in the next chapters.

Now let's talk a little bit about the inner working of the MinHash algorithm.

We now give the steps that the MinHash algorithm follows to extract the MinHash Signature from a document:

1. First of all we extract all the k-character shingles from a document as we described above. For example, for a n-character document we extract $n - k + 1$ shingles and we pass them to our Hash Function, so we transform them to 4-byte integers.
2. We then pass all the hashed shingles through a hash function like:
   hi(x) = (ai * x + bi) % p ,
   where ai and bi are random integers,   p = prime number,
   we will talk about these in the following chapters.
3. We take the minimum integer from the above hash function and we add it in our new Signature.
4. We reiterate through steps 2 and 3, as many times as we want the signature matrix to be in size. That is how many elements we have pre-decided the signature matrix to have. In our case if we want 200 element big signature matrix, we reiterate through steps 2 and 3, 199 times more.

Now it is obvious that we have reduced the space of every document to signatures of 200 elements. We expect that from a "good" hash function that the smallest number has the same probability to appear as the largest number. What we mean by that is that we expect from a good hash algorithm to distribute numbers in buckets equally. We will see in the following chapters that the minhash algorithm is essentially a random selection from the universe of all shingles inside a document. Now, let's continue with the set representation.

3.4 Representation of Sets

To visualize this characteristic Matrix representation of shingles and documents we can imagine a Matrix that has as rows all the shingle Universe that might exist in a document. That means that if we use k = 5 character shingle size, and we use all the English Alphabet as possible characters in the Universe, then our Universe would be 27^5 = 14.348.907 possible shingles. So you can imagine a Matrix with 14.348.907 rows in this case. If we use k = 9 character shingle size and the whole English Alphabet as possible characters as we do in our case, then we would have 27^9 = 76255975e+12 possible shingles, and the same number of rows in our characteristic Matrix.

As columns in the characteristic Matrix, you can imagine the documents themselves. That is, in the first column for example we have Document 1, in the second column Document 2 e.t.c.

Now this characteristic matrix is a Boolean Matrix that has only the values of 0 and 1. It is obvious now that 1 in column "c" and row "r" means that the shingle in that row (that specific shingle) exists in Document "c". Well what do we mean with, that specific shingle is that we have identified all the possible shingles with numbers, and have sorted all these numbers. So all the elements – rows, in our characteristic matrix, you can imagine them as sorted integers – shingles rather than just unordered numbers. This helps to visualize how the characteristic Matrix might be, and how the MInHash algorithm really works. Below we give an example:

Matrix: 3.4.1 Representing 4 Sets – Documents.

Element	*S1*	*S2*	*S3*	*S4*
a	1	0	0	1
b	0	0	1	0
c	0	1	0	1
d	1	0	1	1
e	0	0	1	0
f	0	0	1	0
g	1	0	1	0
h	1	1	0	0

i	0	1	0	1
j	0	1	0	1
k	1	0	0	0
l	1	1	1	1
m	0	0	0	1

The above characteristic Matrix represents 4 Sets – Documents as columns, and a possible Universal set of 13 elements: {a, b, c, d, e, f, g, h, i, j, k, l, m}. So we have:

S1 = {a, d, g, h, k, l}

S2 = {c, h, j, i, l}

S3 = {b, d, e, f, g, l}

S4 = {a, c, d, j, i, l, m}.

It is remarkable to note that the characteristic matrix is likely to be sparse, because it has many 0 elements in the cells, as every document has only a tiny subset of the universe of the shingles. Also we remind that in our case the elements are the k-character shingles, which constitute the rows of the matrix and the documents are the columns. We will examine this later in more detail.

4 The Min-Hash Algorithm

In computer science, Min-Hash is a technique for quickly estimating how similar two sets are. The scheme was invented by Andrei Broder (1997), and initially used in the AltaVista search engine to detect duplicate web pages and eliminate them from search results. [4]

4.1 Min-Hashing

It is known that the minhash algorithm makes extensive use of hashing functions, to compute the minhash signature of the characteristic Matrix. The resulting Matrix that is constructed from the hash functions is composed of a large number of calculation operations (several hundred). Here we will describe how the minhash signature is computed in practice, for a document – set.

Let's try to explain now how the Signature Matrix is created from the characteristic Matrix from scratch. What we essentially do is that we pick N random permutations from the rows of the initial characteristic Matrix and we pick as an element to belong in the Signature of this Set, the first element in the permuted order we will find if we search from top to bottom. Of course this means that this element is in the initial characteristic Matrix and belongs to the Set we examine.

The job of one Hash function means:

1) Permute the order of the initial characteristic Matrix randomly.

   That means all the shingles represented by rows here in the characteristic Matrix from all the Universe will be shuffled randomly.

2) Take the first 1 you will find from the column, as you traverse the column from top to bottom in the newly ordered characteristic Matrix and signify this shingle (number) for possible import in the Signature Set of this column.

3) If the shingle that is about to be imported, (the number) is smaller than the already number – shingle in the Signature Matrix, then replace what already exists in the Signature Matrix with the newly "smaller" shingle.

And that is essentially the whole essence and inner workings of every Min Hash function. We repeat all the above steps 1 – 3 for every hash function we have.

Now how many hash functions we will use and why is explained better in the next chapters, but for now you can assume that it is a user defined number. Also what is the best family of Hash functions to use and why, falls out of the scope of this thesis. You can assume it is safe to use a simple hash function family of type:

hi(x) = (ai * x + b) % p.

where ai, bi are random positive integers from our Universe.

And p is a big prime number. Usually we take p, at least, as big as the maximum number of our Universe. We will talk about this in more detail, in next chapters.

Let's try to give an example that will help a lot in clearing out how the Minhash algorithm works. From the previous characteristic Matrix let's suppose that one hash function gives the following random permutation: {b, j, e, m, i, k, g, a, l, h, f, d, c}.Let' now try to construct the Signature Matrix from the previous characteristic Matrix 3.4.1.

Below is the characteristic Matrix in the permuted order:

Matrix: 4.1.1 Permutation of rows of Matrix 3.4.1

Element	*S1*	*S2*	*S3*	*S4*
b	0	0	1	0
j	0	1	0	1
e	0	0	1	0
m	0	0	0	1
i	0	1	0	1
k	1	0	0	0
g	1	0	1	0
a	1	0	0	1
l	1	1	1	1
h	1	1	0	0

f	0	0	1	0
d	1	0	1	1
c	0	1	0	1

For set S1 we start looking from top to bottom:

Element: b has 0 in S1 set, so we move on to the next row – element, that is element j.

Element: j has also 0 in S1 so we move on to the next row – element, that is element e.

Element: e has 0 too in S1 so we move on to the next row, that is element m.

The next two rows also have 0 in set S1, so we continue with the next row, that is element k.

Row element k has 1 in set S1. So we then check the Signature Matrix. We then essentially exchange all the numbers from h1() to hk() in the Signature Matrix for set S1, that are bigger than the hash values from h1 – hk in this row. We will explain that better in the following chapter when we use the hashed versions of the elements instead of shingles. This is an example to see how the 1st step of the algorithm rolls.

So what we do in essence is:

First we take random permutations among the rows of the characteristic Matrix. That is changing the order of the elements in a set, according to a hash function, and taking the first existing element from every set.

Please note that in practice, even one permutation of the rows in a huge Matrix is strongly discouraged and not possible in terms of efficiency or good programing technique.

4.2 MinHash and Jaccard Similarity Connection

One reasonable question that comes to mind is: Why do we choose the Jaccard Simi larity as a comparison measure between two sets in the Min – Hash Algorithm? It turns out that there is a strong connection between the Min - Hash and the Jaccard Similarity measure and we will explain about it right now.

The general property that holds is that the probability of every hash function that we use in the Min – Hash algorithm for every two random Sets, (let’s denote them as S1 and S2) to give the same number, equals the Jaccard Similarity of these two sets.

It is actually obvious to see that, because whenever these two sets will have 1 in their characteristic Matrix, the very same hash function will give the same result for both of these two Sets, and the probability to give the same results in both of these sets, is the number of rows in the intersection divided by the number of rows of the union of these two Sets, which is the Jaccard Similarity of these two sets. But let's try to explain this with a better and more concrete example.

For two random Sets in our characteristic Matrix:

1) Let's denote as X rows, the rows which have 1's both Sets simultaneously for a specific shingle. That is rows that belong in the intersection of the two Sets.
2) Let's denote as Y rows, the rows that have 1, exclusively one of the two Sets, either S1 or S2. That is when a shingle belongs to exclusively one Set.
3) Let's denote as Z rows, the rows that have 0 both Sets for a specific shingle, or both Sets don't have a specific shingle. These rows are the most common ones as the Boolean characteristic Matrix is sparse.

What we will prove here is that, for two random Sets in our characteristic Matrix, the probability a hash function returns the same result for these two sets, equals the Jaccard Similarity of the very same Sets. Let's assume that there are x rows of type X, and y rows of type Y. We don't care about Z rows because they don't belong in neither the intersection or the union of the two Sets.

Well first of all it is obvious that the Jaccard Similarity of the two Sets equals:

JS(S1, S2) = $\frac{X}{X+Y}$

Probability(h(S1) == h(S2)) = Probability(Characteristic Matrix after the permutation of rows has 1 for S1 AND 1 for S2) = Probability(That the row to be of type X row) = All type X rows divided by all type (X + Y) rows. That is the intersection divided by their Union of the two Sets, which is the Jaccard Similarity of the two Sets = $\frac{X}{X+Y}$ = JS(S1, S2).

4.3 Minhash Signatures

So how do we create the Signature Matrix from the characteristic matrix M in practice?

As mentioned above, every row in the characteristic matrix M represents a unique element from our universe. That means that all rows in the Matrix represent all the unique elements that exist in all documents of our corpus (the Universe).

We mentioned before that constructing the Signature Matrix requires a random number of true permutations in the rows of the characteristic matrix M. These permutations are in the range of several hundred (usually about 200 but we will see in the following chapters how we pick that number). So if we denote each permutation to be a hash function, we then have several hundreds of hash functions [h1, h2, … , h200] for example.

4.4 Creating Minhash Signatures

Picking random permutations of millions or even billions of rows, from a very large characteristic matrix, is infeasible, time consuming and an extremely daunting task. Thus, permuted matrices, as previously mentioned while conceptually appealing, are not implementable at all.

Fortunately, it is possible to simulate the ineffective and time consuming task of random permutations in the rows of matrix M by mapping rows of the characteristic Matrix (shingles), to other rows of the very same characteristic Matrix. Actually, every hash function maps rows to the same range of the rows in the characteristic Matrix (in our case $0, \dots , 2^{31} - 1$). It is important to note here, that what we are actually trying to do is to permute rows in the characteristic Matrix. This means that every hash function gives a specific order of the rows in the characteristic Matrix.

It is important to note the Signature Matrix will have as many columns as the characteristic Matrix, but only the N rows (as many hash functions the user picked).

We conclude to the following MinHash algorithm to build the Signature Matrix from the characteristic Matrix with a good example.

Algorithm 4.1 for creating minhash signatures.

Let’s suppose we have chosen N Hash functions, we then do the following:

1. We calculate h1(r), h2(r), . . . , hN(r) for every row.
2.

    2.1. If there is a 0 in the characteristic Matrix in row r and column c, do nothing.

2.2. Else if there is a 1 in column S, then go to the Signature Matrix and replace in the column S, every row that has bigger number than the corresponding hi, in the characteristic Matrix, with that hi. Otherwise do nothing.

Below is an example of computing the Min-Hash signature of Matrix 3.4.1 with 2 hash functions h1 and h2:

Matrix: 4.1 Representing matrix 3.4.1, with 6 hash functions.

Row	*S1*	*S2*	*S3*	*S4*	*x+1 mod 13*	*3x+1 mod 13*	*5x+1 mod 13*	*7x+1 mod 13*	*9x+1 mod 13*	*11x+1 mod 13*
0	1	0	0	1	1	1	1	1	1	1
1	0	0	1	0	2	4	6	8	10	12
2	0	1	0	1	3	7	11	2	6	10
3	1	0	1	1	4	10	3	9	2	8
4	0	0	1	0	5	0	8	3	11	6
5	0	0	1	0	6	3	0	10	7	4
6	1	0	1	0	7	6	5	4	3	2
7	1	1	0	0	8	9	10	11	12	0
8	0	1	0	1	9	12	2	5	8	11
9	0	1	0	1	10	2	7	12	4	9
10	1	0	0	0	11	5	12	6	0	7
11	1	1	1	1	12	8	4	0	9	5
12	0	0	0	1	0	11	9	7	5	3

Let's try to explain the above example. The matrix above represents matrix 3.4.2 with some additional data. Instead of shingles, we use integer numbers between: [0, …, 12] (These numbers denote 13 distinct elements [a,…, m]) .

We have picked the following 6 hash-functions: h1(x) = (x+1) mod 13, h2(x) = (3x+1) mod 13, h3(x) = (5x+1) mod 13, h4(x) = (7x+1) mod 13, h5(x) = (9x+1) mod

13, and h6(x) = (11x+1) mod 13. The results of these 6 hash functions are shown in the last columns of the characteristic Matrix. Notice that we picked number 13 as a divisor, for two reasons. First because it is a prime number, and secondly because it is next bigger number of all the elements – shingles in the Universe. This means that it covers all the rows in the characteristic Matrix for every Hash Function. Generally you can expect collisions to happen when 2 rows hash to the same integer value.

Let’s try now to compute the signature matrix, for this example, from the above algorithm. Initially the signature matrix consists of all ∞’s, as denoted in the algorithm above:

Sig Matrix 1. Representing the initial stage of the signature Matrix.

Hash function	*S1*	*S2*	*S3*	*S4*
h1	∞	∞	∞	∞
h2	∞	∞	∞	∞
h3	∞	∞	∞	∞
h4	∞	∞	∞	∞
h5	∞	∞	∞	∞
h6	∞	∞	∞	∞

First, we start with row 0 from the Matrix 4.1. We can see that all the hash functions: h1(0), h2(0), h3(0), h4(0), h5(0) and h6(0) have 1 in their cells. The row number 0, has 1s only in the column sets: S4 and S1. Therefore only these are subject to change. We can clearly see that 1 is less than ∞, so we change both sets S1 and S4. So the signature matrix becomes:

Sig Matrix 1. Representing the initial stage of the signature Matrix.

Hash function	*S1*	*S2*	*S3*	*S4*
h1	1	∞	∞	1
h2	1	∞	∞	1
h3	1	∞	∞	1
h4	1	∞	∞	1
h5	1	∞	∞	1

h6	1	∞	∞	1

Let’s proceed with row 1 in matrix 4.1. Only column S3 has 1, and the corresponding hash values are h1(1) = 2, h2(1) = 4, h3(1) = 6, h4(1) = 8, h5(1) = 10, h6(1) = 12. So according to the algorithm, the signatures become: SIG(1, 3) to 2, SIG(2, 3) to 4, SIG(3, 3) to 6, SIG(4, 3) to 8, SIG(5, 3) to 10 and SIG(6, 3) to 12. SIG(i,3) = ∞, for all i=1,2,…, 6 so all signatures change in S3. All the other elements the Signature Matrix are not subjected to any changes because there is 0’s in the corresponding characteristic Matrix. So we have the following:

Signature Matrix 2.

Hash function	*S1*	*S2*	*S3*	*S4*
h1	1	∞	2	1
h2	1	∞	4	1
h3	1	∞	6	1
h4	1	∞	8	1
h5	1	∞	10	1
h6	1	∞	12	1

The row numbered 2 in the initial matrix 4.1, has 1s in column sets: S2 and S4. The hash values for row 2 are: h1(2) = 3, h2(2) = 7, h3(2) = 11, h4(2) = 2, h5(2) = 6, h6(2) = 10.

There was a possibility to change column S4 in the Signature Matrix as there is a 1 in the characteristic Matrix, but because column S4 in the Signature Matrix is [1, 1, 1, 1, 1, 1], are each less than the corresponding hash values [3, 7, 11, 2, 6, 10], so they remain unchanged. Since column S2 has the original initialization of ∞, we substitute them by the new hash values [3, 7, 11, 2, 6, 10] that resulted, as follows:

Signature Matrix 3.

Hash function	*S1*	*S2*	*S3*	*S4*
h1	1	3	2	1
h2	1	7	4	1
h3	1	11	6	1
h4	1	2	8	1
h5	1	6	10	1
h6	1	10	12	1

We move on to the next row from our initial matrix, (row 3). All columns but S2 here, have 1s, and the hash-values from our hash functions are h1(3) = 4, h2(3) = 10, h3(3) = 3, h4(3) = 9, h5(3) = 2, h6(3) = 8. If we examine the Signature matrix 3 closely, we will notice that the value h1(3) = 4 > what is already in the signature matrix, for columns S1, S3 and S4 for h1 hash function, so the first row of the signature matrix doesn't change. We continue traversing the other rows of the signature matrix having in mind the corresponding hash values. Again h2(3) = 10 is bigger than any of S1, S3 and S4 corresponding values of the signature matrix, so they don't change. Now if we move on to the next row of the signature matrix [1, 11, 6, 1], we can see that only S3 can change, because h3(3) = 3, and is bigger than S3. So we will change that element from S3 to 3. Accordingly, we check the following rows of the signature matrix:

- h4(3) = 9 > SigM[h4][S1, S3, S4] : So they remain the same.
- h5(3) = 2, which is bigger than SigM[h5][S1] and SigM[h5][S4], so these two remain the same, but it is smaller than SigM[h5][S3], so SigM[h5][S3] changes to 2.
- h6(3) = 8. 8 is only lower than SigM[h6][S3]. So SigM[h6][S3] becomes 8 from 12 before.

So the resulting signature matrix, becomes the following:

Signature Matrix 4.

Hash function	*S1*	*S2*	*S3*	*S4*
h1	1	3	2	1
h2	1	7	4	1
h3	1	11	3	1
h4	1	2	8	1
h5	1	6	2	1
h6	1	10	8	1

Notice that S2 column wasn’t part of the comparisons, because it had 0 at the initial matrix in row numbered 3. Let’s continue with the other rows.

In row number 4, of the original matrix 4.1, we notice that only S3 column has 1. So, only S3 column is likely to change, if any changes occur. Let’s examine the hash values derived from the hash functions : h1(4) = 5, h2(4) = 0, h3(4) = 8, h4(4) = 3, h5(4) = 11, h6(4) = 6. Following the previous tactic, for the column S3 [2, 4, 3, 8, 2, 8], we have the following:

- h1(4) > SigM[h1][S3] => 5 > 2 . SigM[h1][S3] remains the same.
- h2(4) < SigM[h2][S3] => 0 < 4 . SigM[h2][S3] changes to 0.
- h3(4) > SigM[h3][S3] => 8 > 3 . SigM[h3][S3] remains the same.
- h4(4) < SigM[h4][S3] => 3 < 8 . SigM[h4][S3] changes to 3.
- h5(4) > SigM[h5][S3] => 11 > 2 . SigM[h5][S3] remains the same.
- h6(4) < SigM[h6][S3] => 6 < 8 . SigM[h6][S3] changes to 6.

The new signature Matrix becomes:

Signature Matrix 5.

Hash function	*S1*	*S2*	*S3*	*S4*
h1	1	3	2	1
h2	1	7	0	1
h3	1	11	3	1
h4	1	2	3	1

h5	1	6	2	1
h6	1	10	6	1

We move on to the next row, row numbered 5 with the same pattern. We can clearly see that row number 5, has also 1 only in column S3. So, as seen before, that is the only column that might be subjected to change. The hash values which derive from the corresponding hash functions are: h1(5) = 6, h2(5) = 3, h3(5) = 0, h4(5) = 10, h5(5) = 7, h6(5) = 4. Same as before, for the column S3 [2, 0, 3, 3, 2, 6] from the previous signature matrix, we have the following:

- SigM[h1][S3] = 2 < h1(5) = 6 => 2 < 6 . SigM[h1][S3] remains the same.
- SigM[h2][S3] = 0 < h2(5) = 3 => 0 < 3 . SigM[h2][S3] remains the same.
- SigM[h3][S3] = 3 > h3(5) = 0 => 3 > 0 . SigM[h3][S3] changes to 0.
- SigM[h4][S3] = 3 < h4(5) = 10 => 3 < 10 . SigM[h4][S3] remains the same.
- SigM[h5][S3] = 2 < h5(5) = 7 => 2 < 7 . SigM[h5][S3] remains the same.
- SigM[h6][S3] = 6 > h6(5) = 4 => 6 > 4 . SigM[h6][S3] changes to 4.

The new signature Matrix becomes:

Signature Matrix 6.

Hash function	*S1*	*S2*	*S3*	*S4*
h1	1	3	2	1
h2	1	7	0	1
h3	1	11	0	1
h4	1	2	3	1
h5	1	6	2	1
h6	1	10	4	1

Notice that S1, S2 and S4 columns were not part of the equation, because they had 0 at the initial matrix, in row number 5. Therefore we didn’t bother checking them at all. Let’s continue with the next row, row number 6.

In row number 6, of the original matrix 4.1, we notice that only S1 and S3 columns have 1. So, only S1 and S3 columns are likely to change, if any changes occur. Let’s examine the hash values derived from the hash functions :

h1(6) = 7, h2(6) = 6, h3(6) = 5, h4(6) = 4, h5(6) = 3, h6(6) = 2. Following from the previous signature matrix for columns S1[1, 1, 1, 1, 1, 1], and S3[2, 0, 0, 3, 2, 4], we have the following for S1:

- SigM[h1][S1] = 1 < h1(6) = 7 => 1 < 7 . SigM[h1][S1] remains the same.
- SigM[h2][S1] = 1 < h2(6) = 6 => 1 < 6 . SigM[h2][S1] remains the same.
- SigM[h3][S1] = 1 < h3(6) = 5 => 1 < 5 . SigM[h3][S1] remains the same.
- SigM[h4][S1] = 1 < h4(6) = 4 => 1 < 4 . SigM[h4][S1] remains the same.
- SigM[h5][S1] = 1 < h5(6) = 3 => 1 < 3 . SigM[h5][S1] remains the same.
- SigM[h6][S1] = 1 < h6(6) = 2 => 1 < 2 . SigM[h6][S1] remains the same.

And for S3:

- SigM[h1][S3] = 2 < h1(6) = 7 => 2 < 7 . SigM[h1][S3] remains the same.
- SigM[h2][S3] = 0 < h2(6) = 6 => 0 < 6 . SigM[h2][S3] remains the same.
- SigM[h3][S3] = 0 < h3(6) = 5 => 0 < 5 . SigM[h3][S3] remains the same.
- SigM[h4][S3] = 3 < h4(6) = 4 => 3 < 4 . SigM[h4][S3] remains the same.
- SigM[h5][S3] = 2 < h5(6) = 3 => 2 < 3 . SigM[h5][S3] remains the same.
- SigM[h6][S3] = 4 > h6(6) = 2 => 4 > 2 . SigM[h6][S3] changes to 2.

So the new signature Matrix results:

Signature Matrix 7.

Hash function	*S1*	*S2*	*S3*	*S4*
h1	1	3	2	1
h2	1	7	0	1
h3	1	11	0	1
h4	1	2	3	1
h5	1	6	2	1
h6	1	10	2	1

Notice that S2 and S4 columns had 0 at the initial matrix, in row number 6. So we didn't check them at all, and they remained unchanged. Let's continue with the next row, row number 7.

In row number 7, of the original matrix 4.1, we can see that only S1 and S2 columns have 1's. So, only S1 and S2 columns are likely to change, if any changes happen. Let's examine the hash values derived from the hash functions :

h1(7) = 8, h2(7) = 9, h3(7) = 10, h4(7) = 11, h5(7) = 12, h6(7) = 0 or

[8, 9, 10, 11, 12, 0]. Following from the previous signature matrix for columns

S1[1, 1, 1, 1, 1, 1], and S2[3, 7, 11, 2, 6, 10], we have the following:

For S1:

- SigM[h1][S1] = 1 < h1(7) = 8 => 1 < 8 . SigM[h1][S1] remains the same.
- SigM[h2][S1] = 1 < h2(7) = 9 => 1 < 9 . SigM[h2][S1] remains the same.
- SigM[h3][S1] = 1 < h3(7) = 10 => 1 < 10 . SigM[h3][S1] remains the same.
- SigM[h4][S1] = 1 < h4(7) = 11 => 1 < 11 . SigM[h4][S1] remains the same.
- SigM[h5][S1] = 1 < h5(7) = 12 => 1 < 12 . SigM[h5][S1] remains the same.
- SigM[h6][S1] = 1 > h6(7) = 0 => 1 > 0 . SigM[h6][S1] changes to 0.

And for S2:

- SigM[h1][S2] = 3 < h1(7) = 8 => 3 < 8 . SigM[h1][S2] remains the same.
- SigM[h2][S2] = 7 < h2(7) = 9 => 7 < 9 . SigM[h2][S2] remains the same.
- SigM[h3][S2] = 11 > h3(7) = 10 => 11 > 10 . SigM[h3][S2] changes to 10.
- SigM[h4][S2] = 2 < h4(7) = 11 => 2 < 11 . SigM[h4][S2] remains the same.
- SigM[h5][S2] = 6 < h5(7) = 12 => 6 < 12 . SigM[h5][S2] remains the same.
- SigM[h6][S2] = 10 > h6(7) = 0 => 10 > 0 . SigM[h6][S2] changes to 0.

So the new signature Matrix is:

Signature Matrix 7.

Hash function	*S1*	*S2*	*S3*	*S4*
h1	1	3	2	1
h2	1	7	0	1
h3	1	10	0	1
h4	1	2	3	1
h5	1	6	2	1
h6	0	0	2	1

Note that S3 and S4 columns had 0 in the initial matrix, in row number 7, so we didn't consider them in the comparisons, therefore they remained unchanged. Let's continue with the next row, row number 8.

In row number 8, of the original matrix 4.1, we can see that only S2 and S4 have 1's in their columns. So, only S2 and S4 columns are likely to change. Let's examine the hash values derived from the hash functions:

h1(8) = 9, h2(8) = 12, h3(8) = 2, h4(8) = 5, h5(8) = 8, h6(8) = 11 or

[9, 12, 2, 5, 8, 11]. Following from the previous signature matrix for columns

S2[3, 7, 10, 2, 6, 0], and S4[1, 1, 1, 1, 1, 1], we have the following:

For S2:

- SigM[h1][S2] = 3 < h1(8) = 9 => 3 < 9 . SigM[h1][S2] remains the same.
- SigM[h2][S2] = 7 < h2(8) = 12 => 7 < 12 . SigM[h2][S2] remains the same.
- SigM[h3][S2] = 10 > h3(8) = 2 => 10 > 2 . SigM[h3][S2] changes to 2.
- SigM[h4][S2] = 2 < h4(8) = 5 => 2 < 5 . SigM[h4][S2] remains the same.
- SigM[h5][S2] = 6 < h5(8) = 8 => 6 < 8 . SigM[h5][S2] remains the same.
- SigM[h6][S2] = 0 < h6(8) = 11 => 0 < 11 . SigM[h6][S2] remains the same.

For S4:

- SigM[h1][S4] = 1 < h1(8) = 9 => 1 < 9 . SigM[h1][S4] remains the same.
- SigM[h2][S4] = 1 < h2(8) = 12 => 1 < 12 . SigM[h2][S4] remains the same.
- SigM[h3][S4] = 1 < h3(8) = 2 => 1 < 2 . SigM[h3][S4] remains the same.

- SigM[h4][S4] = 1 < h4(8) = 5 => 1 < 5 . SigM[h4][S4] remains the same.
- SigM[h5][S4] = 1 < h5(8) = 8 => 1 < 8 . SigM[h5][S4] remains the same.
- SigM[h6][S4] = 1 < h6(8) = 11 => 1 < 11 . SigM[h6][S4] remains the same.

So S4 remains the same because all elements of column S4 in the Signature Matrix 7, are smaller than the corresponding hash functions h1 to h6. So the new signature matrix 8, becomes:

Signature Matrix 8.

Hash function	*S1*	*S2*	*S3*	*S4*
h1	1	3	2	1
h2	1	7	0	1
h3	1	2	0	1
h4	1	2	3	1
h5	1	6	2	1
h6	0	0	2	1

S1 and S3 columns had 0 at the initial matrix, in row 8, so we didn’t check them at all, therefore they remained unchanged. Let’s continue with the next row from our initial matrix, row number 9.

In row number 9, of the original matrix 4.1, we can see again, that only S2 and S4 have 1’s in their columns. Therefore, only S2 and S4 columns are likely to change. Let’s examine the hash values derived from the corresponding hash functions:

h1(9) = 10, h2(9) = 2, h3(9) = 7, h4(9) = 12, h5(9) = 4, h6(9) = 9 or

[10, 2, 7, 12, 4, 9]. Following from the previous signature matrix for columns

S2[3, 7, 2, 2, 6, 0], and S4[1, 1, 1, 1, 1, 1], we have the following:

For S2:

- SigM[h1][S2] = 3 < h1(9) = 10 => 3 < 10 . SigM[h1][S2] remains the same.
- SigM[h2][S2] = 7 > h2(9) = 2 => 7 > 2 . SigM[h2][S2] changes to 2.
- SigM[h3][S2] = 2 < h3(9) = 7 => 2 < 7 . SigM[h3][S2] remains the same.
- SigM[h4][S2] = 2 < h4(9) = 12 => 2 < 12 . SigM[h4][S2] remains the same.

- SigM[h5][S2] = 6 > h5(9) = 4 => 6 > 4 . SigM[h5][S2] changes to 4.
- SigM[h6][S2] = 0 < h6(9) = 9 => 0 < 9 . SigM[h6][S2] remains the same.

For S4:

- SigM[h1][S4] = 1 < h1(9) = 10 => 1 < 10 . SigM[h1][S4] remains the same.
- SigM[h2][S4] = 1 < h2(9) = 2 => 1 < 2 . SigM[h2][S4] remains the same.
- SigM[h3][S4] = 1 < h3(9) = 7 => 1 < 7 . SigM[h3][S4] remains the same.
- SigM[h4][S4] = 1 < h4(9) = 12 => 1 < 12 . SigM[h4][S4] remains the same.
- SigM[h5][S4] = 1 < h5(9) = 4 => 1 < 4 . SigM[h5][S4] remains the same.
- SigM[h6][S4] = 1 < h6(9) = 9 => 1 < 9 . SigM[h6][S4] remains the same.

So S4 still remains the same, because all elements of column S4 in the Signature Matrix 8, are smaller than the corresponding hash functions from [h1 to h6]. So the new signature matrix 9, becomes:

Signature Matrix 9.

Hash function	*S1*	*S2*	*S3*	*S4*
h1	1	3	2	1
h2	1	2	0	1
h3	1	2	0	1
h4	1	2	3	1
h5	1	4	2	1
h6	0	0	2	1

S1 and S3 columns had 0 at the initial matrix, in row 9, so they are ignored. Therefore they remained unchanged. We continue with the next row from our initial matrix, row number 10.

In row number 10, of the original matrix 4.1, we see that only S1 column has 1. Therefore, only this column is likely to change. Let’s examine the hash values derived from the corresponding hash functions:

h1(10) = 11, h2(10) = 5, h3(10) = 12, h4(10) = 6, h5(10) = 0, h6(10) = 7 or

[11, 5, 12, 6, 0, 7]. Following from the previous signature matrix the column

S1[1, 1, 1, 1, 1, 0], we have the following :

For S1:

- SigM[h1][S1] = 1 < h1(10) = 11 => 1 < 11 . SigM[h1][S1] remains the same.
- SigM[h2][S1] = 1 < h2(10) = 5 => 1 < 5 . SigM[h2][S1] remains the same.
- SigM[h3][S1] = 1 < h3(10) = 12 => 1 < 12 . SigM[h3][S1] remains the same.
- SigM[h4][S1] = 1 < h4(10) = 6 => 1 < 6 . SigM[h4][S1] remains the same.
- SigM[h5][S1] = 1 > h5(10) = 0 => 1 > 0 . SigM[h5][S1] changes to 0.
- SigM[h6][S1] = 0 < h6(10) = 7 => 0 < 7 . SigM[h6][S1] remains the same.

So the new signature matrix, after calculating the hash functions and comparing S1 column becomes:

Signature Matrix 10.

Hash function	*S1*	*S2*	*S3*	*S4*
h1	1	3	2	1
h2	1	2	0	1
h3	1	2	0	1
h4	1	2	3	1
h5	0	4	2	1
h6	0	0	2	1

All but S1 columns had 0 at the initial matrix, in row 10, so they were ignored. Therefore they remained unchanged. We continue with the next row from our initial matrix, row number 11.

In row number 11, of the original matrix, we observe that all columns have 1 in their cells. This means that all columns are likely to change. Let's now examine the hash values derived from the corresponding hash functions:

h1(11) = 12, h2(11) = 8, h3(11) = 4, h4(11) = 0, h5(11) = 9, h6(11) = 5 or

[12, 8, 4, 0, 9, 5]. Following from the previous signature matrix, for the column

S1[1, 1, 1, 1, 0, 0], we have the following :

S1:

- SigM[h1][S1] = 1 < h1(11) = 12 => 1 < 12 . SigM[h1][S1] remains the same.
- SigM[h2][S1] = 1 < h2(11) = 8 => 1 < 8 . SigM[h2][S1] remains the same.
- SigM[h3][S1] = 1 < h3(11) = 4 => 1 < 4 . SigM[h3][S1] remains the same.
- SigM[h4][S1] = 1 > h4(11) = 0 => 1 > 0 . SigM[h4][S1] changes to 0.
- SigM[h5][S1] = 0 < h5(11) = 9 => 0 < 9 . SigM[h5][S1] remains the same.
- SigM[h6][S1] = 0 < h6(11) = 5 => 0 < 5 . SigM[h6][S1] remains the same.

Again we have H[12, 8, 4, 0, 9, 5]

And S2[3, 2, 2, 2, 4, 0]

For S2:

- SigM[h1][S2] = 3 < h1(11) = 12 => 3 < 12 . SigM[h1][S2] remains the same.
- SigM[h2][S2] = 2 < h2(11) = 8 => 2 < 8 . SigM[h2][S2] remains the same.
- SigM[h3][S2] = 2 < h3(11) = 4 => 2 < 4 . SigM[h3][S2] remains the same.
- SigM[h4][S2] = 2 > h4(11) = 0 => 2 > 0 . SigM[h4][S2] changes to 0.
- SigM[h5][S2] = 4 < h5(11) = 9 => 4 < 9 . SigM[h5][S2] remains the same.
- SigM[h6][S2] = 0 < h6(11) = 5 => 0 < 5 . SigM[h6][S2] remains the same.

H[12, 8, 4, 0, 9, 5]

S3[2, 0, 0, 3, 2, 2]

For S3, we have:

- SigM[h1][S3] = 2 < h1(11) = 12 => 2 < 12 . SigM[h1][S3] remains the same.
- SigM[h2][S3] = 0 < h2(11) = 8 => 0 < 8 . SigM[h2][S3] remains the same.
- SigM[h3][S3] = 0 < h3(11) = 4 => 0 < 4 . SigM[h3][S3] remains the same.
- SigM[h4][S3] = 3 > h4(11) = 0 => 3 > 0 . SigM[h4][S3] changes to 0.
- SigM[h5][S3] = 2 < h5(11) = 9 => 2 < 9 . SigM[h5][S3] remains the same.
- SigM[h6][S3] = 2 < h6(11) = 5 => 2 < 5 . SigM[h6][S3] remains the same.

H[12, 8, 4, 0, 9, 5]

S4[1, 1, 1, 1, 1, 1]

And finaly for S4, we have:

- SigM[h1][S4] = 1 < h1(11) = 12 => 1 < 12 . SigM[h1][S4] remains the same.
- SigM[h2][S4] = 1 < h2(11) = 8 => 1 < 8 . SigM[h2][S4] remains the same.
- SigM[h3][S4] = 1 < h3(11) = 4 => 1 < 4 . SigM[h3][S4] remains the same.
- SigM[h4][S4] = 1 > h4(11) = 0 => 1 > 0 . SigM[h4][S4] changes to 0.
- SigM[h5][S4] = 1 < h5(11) = 9 => 1 < 9 . SigM[h5][S4] remains the same.
- SigM[h6][S4] = 1 < h6(11) = 5 => 1 < 5 . SigM[h6][S4] remains the same.

So the new Signature matrix, after reviewing all columns and hash values from the hash functions that came up, becomes:

Signature Matrix 11.

Hash function	*S1*	*S2*	*S3*	*S4*
h1	1	3	2	1
h2	1	2	0	1
h3	1	2	0	1
h4	0	0	0	0
h5	0	4	2	1
h6	0	0	2	1

In this row case, all columns changed their state, because all sets: S1, S2, S3, S4 had 1 in that row: row number 11, and all these sets had at least 1 element (in our case exactly 1) bigger than the corresponding hash value that derived from the corresponding hash function.

Let's move on now to the last row, row number 12 from our original matrix. In row number 12, of the original matrix, we can see that all columns except S4 have 0's in their cells. This means that, only S4 column is likely to change. Let's now examine the hash values derived from the corresponding hash functions:

h1(12) = 0, h2(12) = 11, h3(12) = 9, h4(12) = 7, h5(12) = 5, h6(12) = 3 or [0, 11, 9, 7, 5, 3]. Following from the previous signature matrix 10, for column S4[1, 1, 1, 0, 1, 1], we have the following :

S4:

- SigM[h1][S4] = 1 > h1(12) = 0 => 1 > 0 . SigM[h1][S4] changes to 0.
- SigM[h2][S4] = 1 < h2(12) = 11 => 1 < 11 . SigM[h2][S4] remains the same.
- SigM[h3][S4] = 1 < h3(12) = 9 => 1 < 9 . SigM[h3][S4] remains the same.
- SigM[h4][S4] = 0 < h4(12) = 7 => 0 < 7 . SigM[h4][S4] remains the same.
- SigM[h5][S4] = 1 < h5(12) = 5 => 1 < 5 . SigM[h5][S4] remains the same.
- SigM[h6][S4] = 1 < h6(12) = 3 => 1 < 3 . SigM[h6][S4] remains the same.

So the final Signature matrix, after traversing all rows one by one, and applying the previous algorithm 4.1, we saw at the beginning for creating the minhash signatures, becomes the following:

Final Signature Matrix.

Hash function	*S1*	*S2*	*S3*	*S4*
h1	1	3	2	0
h2	1	2	0	1
h3	1	2	0	1
h4	0	0	0	0
h5	0	4	2	1
h6	0	0	2	1

What we have achieved here is that we can make an estimation of the true Jaccard Similarity of Sets: S1, S2, S3 and S4 from our Final Signature Matrix with high probability! For example we notice that, from our final signature matrix column S1 and S4 have Jaccard Similarity JS(S1, S4) = (S1 intersect S4) / (S1 union S4, (That is all the elements that belong to the Union of the 2 sets)). So JS(S1, S4) $= \frac{2}{5} = 0.4$. If we see closely at the initial matrix 4.1, we can derive that the true Jaccard Similarity of the sets S1 and S4 is $\frac{3}{10} = 0.3$. This number is very close to our estimation 0.4.

Although this example is very small to make a good prediction, the rule is that the more hash functions you take as a parameter, the better the estimation. If we extend it, the signature columns for S1 and S2 that derive from our final signature matrix is 0 (true similarity $\frac{2}{9}$ = 0.222… too small), while the signatures of S2 and S3 estimate also to 0, their true Jaccard similarity is $\frac{1}{10}$ = 0.1.

5 Locality Sensitive Hashing

Locality-sensitive hashing (LSH) reduces the dimensionality of high-dimensional data. LSH hashes input items so that similar items map to the same “buckets” with high probability. LSH differs from conventional and cryptographic hash functions because it aims to maximize the probability of a “collision” for similar items. [5] LSH has common use in clustering and Nearest Neighbor (N-N) search problems.

5.1 LSH and Documents

What we have achieved so far, is that we have managed to significantly reduce the space needed to store the elements, (in our case shingles which have been hashed to integer numbers) needed for a document. But as we said earlier in previous chapters, even if we managed to significantly reduce the space needed for every document, the pairwise Jaccard Similarities of all the documents that are being compared is very large, even for a small amount of documents, making this approach of finding the greatest similarity pairs inefficient. For example if we consider 10 documents, and do the number of comparisons naively, that would be:

$$C\binom{10}{2} = \frac{10!}{2! \times (10-2)!} = \frac{10 \times (10-1) \times (10-2)!}{2 \times (10-2)!} = \frac{10 \times 9 \times 8!}{2 \times 8!} = \frac{10 \times 9}{2} = \frac{90}{2} = 45$$

comparisons! Imagine for 1 million or a billion documents, how many comparisons we would have! The general rule of the number of comparisons in a corpus of N documents is :

$$C\binom{N}{2} = \frac{N!}{2! \times (N-2)!} = \frac{N \times (N-1) \times (N-2)!}{2 \times (N-2)!} = \frac{N \times (N-1)}{2}$$

Number which becomes very large, even for a “few” documents.

Although we can use parallelism to reduce the time needed to compare sets of hashed shingles; this mitigation of the time needed to make all these comparisons possible, is infinitesimal compared to the huge number of comparisons needed to examine every tuple of sets in the corpus. However, there is something we can do, to find very efficiently the most significant – greatest pairs that are possible to be similar. There is a method that examines only pairs of Sets that might be similar to each other (those and only those), thereby avoiding all other unnecessary pairs. This method is named Locality Sensitive Hashing or LSH for short and has many applications in nearest neighbor search problems. Because Locality Sensitive Hashing is a general form of dimensionality reduction scheme, we will focus our attention to a more particular structure of LSH, that is ideally used in the Min-Hash algorithm for finding near duplicates.

5.2 LSH and Min-Hash signatures

For the purpose of LSH we use Bands and Buckets. We will explain shortly how we use the Band and Bucket scheme. For all Bands we determine a specific number of buckets according to a mathematical formula which we will see shortly after. So we have for every bucket, a fixed number of items or elements that can be stored inside the bucket.

Let’s try to explain how the LSH technique works and how we can use it in our case to efficiently find the most identical documents. First of all we must have created the Min-Hash signature matrix from the initial characteristic Matrix, as described in the previous chapters. Then we divide the Signature Matrix to B regions – Bands we will call them, of r rows contained in each Band. Obviously it must hold:

- $B \times r =$ #rows in the Signature Matrix.

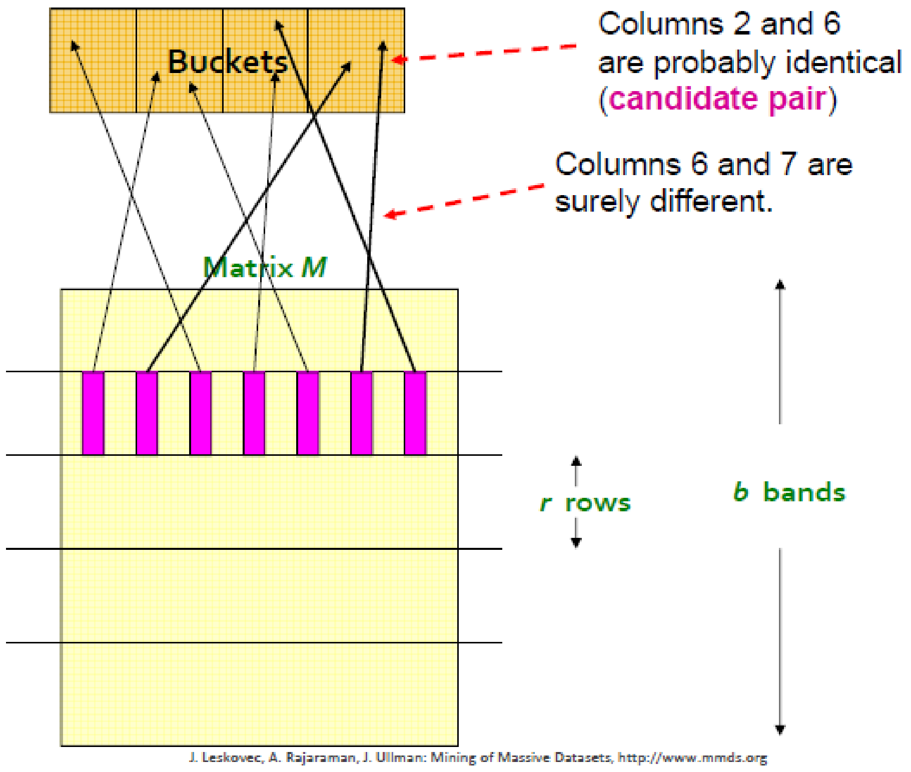
Then we Hash every Band into a big number of buckets, we will refer to it as Bucket Array List. Usually when we say big number of buckets, what we mean is at least as big as all the elements in the Universe. This in our case is: $2^{32} - 1$. So what we expect is that, only those Sets that hashed to the same bucket become the candidate pairs, and only these will be examined for similarity. Hence, we expect that only the most similar items will hash to the same bucket, and the dissimilar ones will not.

Of course there might be false positives, that is dissimilar pairs of Sets that hashed to the same bucket, and false negatives (similar pairs that didn't hashed to the same buckets in all the Bucket Array Lists). But we expect that all false positives and false negatives will be just a tiny percentage of the true positive and true negative pairs accordingly.

So as we said earlier, we take every Band and hash it to a big Bucket Array List. That means we have as many Bucket Array Lists as the number of Bands. We can use a different Hash Function for every Band that maps subsets of the signature Matrix to a big number of buckets but that is not essential. What is critical here is to use a good Hash function that maps Band Sets to a big number of Buckets. This actually ensures us that only two quite similar Sets will hash to the same bucket number and these Sets will be considered similar and later being examined if they are truly similar.

The following figure 5.1 depicts the implementation of the Banding technique in the minhash signature. It depicts the Signature Matrix divided in B bands of r rows each band.

Figure 5.1: Hashing Bands

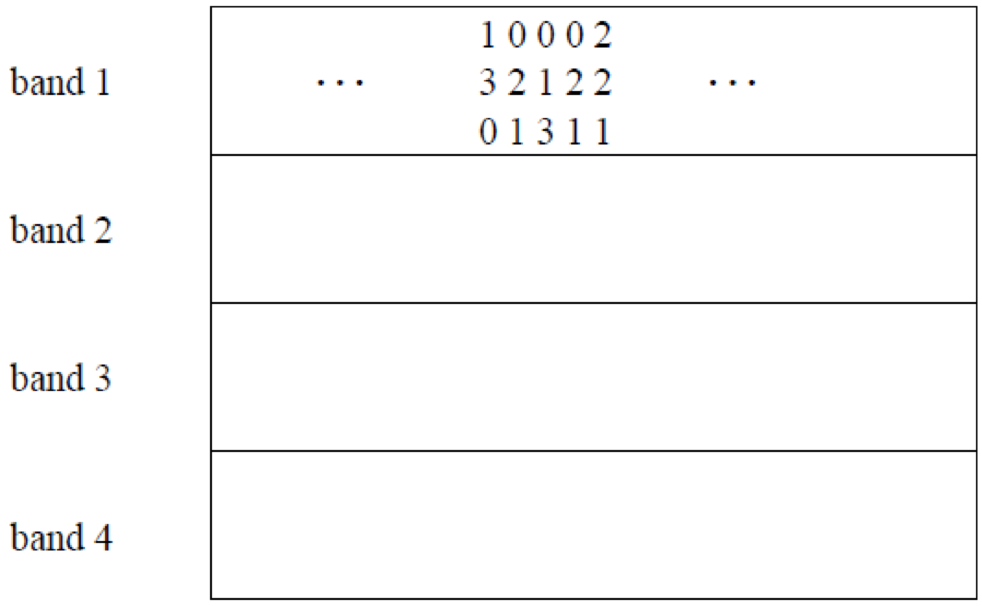

Buckets
Columns 2 and 6 are probably identical (candidate pair)
Columns 6 and 7 are surely different.
Matrix M
r rows
b bands
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Let’s take a closer look at this LSH Banding technique to get a better understanding on how it works with the following example.

Figure 5.2 that follows shows a signature Matrix of 12 rows in total, that has been divided by the Banding technique into 4 Bands of 3 rows each. If we look closer at Band 1, that subset of the signature Matrix, it is easy to notice that columns 2 and 4 have the same sequence of numbers: {0, 2, 1}, so we expect that columns 2 and 4 to hash to the same bucket number in the Bucket Array List 1 that corresponds to Band 1. According to what was said above, we can expect that there will not be any other pair of columns that will hash to the same bucket number in this Bucket Array List. This can be better assured depending on how many buckets we can afford in the Bucket Array List and the capability of the hash function we use to make a good sharing of a set in a given number of buckets.

According to the scheme in Figure 5.2 it is possible for columns 1 and 2 to hash to the same bucket in another Bucket Array List other than the first, if they have the same sequence of numbers for example. So we note that there are 3 other chances for every pair of documents that didn’t hash to the same bucket number in the first Bucket Array List, to hash in some bucket in another Bucket Array List. Be noted that there can be accidental collisions but we expect this to be in minimum levels. Figure 5.2 below shows this in detail:

Figure 5.2 : Signature Matrix division consisting of 4 Bands with 3 rows each[1]



What we are more interested here is to minimize the false negatives, that is truly similar pairs that never hashed to the same bucket number. So what we will expect and is more important to us from the Banding strategy technique is to avoid false negatives from our results.

5.3 Banding Technique

Now let's try to analyze the Banding strategy with an example. Let's suppose that for two Sets S1 and S2 in the characteristic Matrix there is true Jaccard Similarity: s. Also recall from Section 4.2 that the probability to be equal in any row in the Signature Matrix for these two sets is the same as the Jaccard Similarity: s.

We will now calculate the probability for these two sets to become a candidate pair for comparison. According to the Banding Strategy, recall that two Sets become a candidate pair only if any of the Bands from the Signature Matrix hashes to the same bucket, or equivalently if there is a bucket in any of the Bucket Array Lists that has two or more sets hashed inside it. Let's try to find the probability two Sets, hash to same bucket, in any bucket of the bucket Array Lists. That means, calculating the probability that two Sets become a candidate pair. Below we give some probabilities for two Sets S1 and S2, for any band B with r number of rows per band:

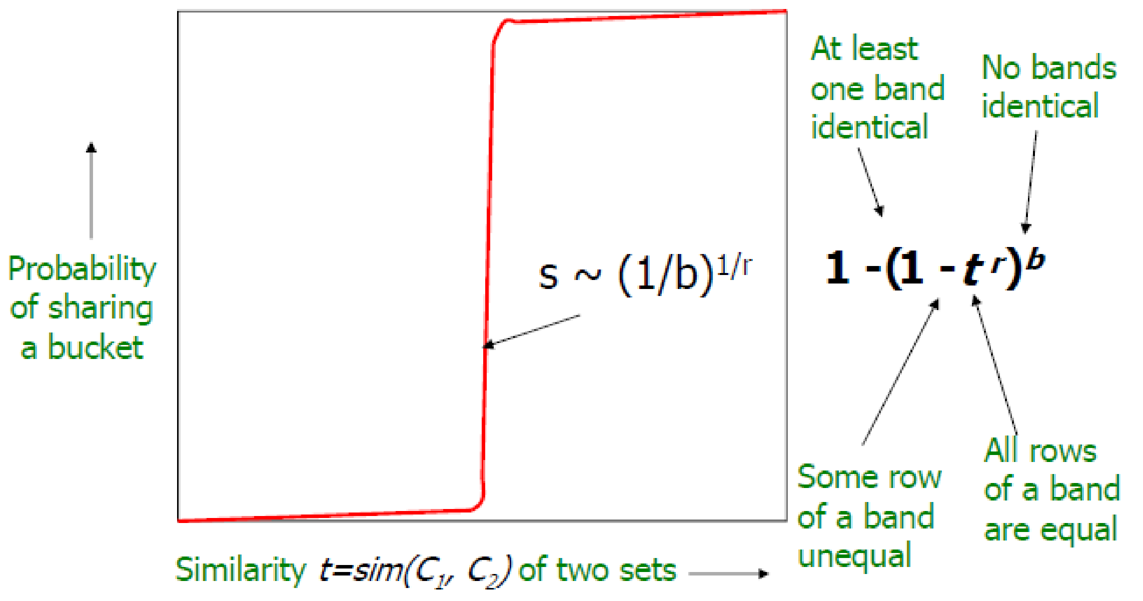
1) The probability of agreement in one row of a Band between S1 and S2 is s.
2) The probability all rows agree in one Band is: s^r.
3) The probability at least one row disagrees in one Band is: $1 - s^r$.
4) The probability at least one row disagrees in one Band for all Bands is:

   $(1 - s^r)^b$. That means there is not a single Band that agrees in all rows.
5) The probability that there is at least one band that agrees in all rows is:

   $1 - (1 - s^r)^b$.

If we try to examine it a little closer, we will see that this function resembles a lot to an S – Curve function, where there exists a big leap somewhere in the function that rises suddenly in a small width of x'x. From the Figure 5.3 that follows, we will see that this leap is found for s = 0.5. We can estimate the similarity threshold with a function of b Bands and r rows per band.

Figure 5.3 : b-bands of r-Rows


Probability of sharing a bucket
$s \sim (1/b)^{1/r}$
At least one band identical
No bands identical
$1 - (1 - t^r)^b$
Some row of a band unequal
All rows of a band are equal
Similarity $t = sim(C_1, C_2)$ of two sets

So what is the approximation function with b and r parameters we can use to calculate this S – Curve?

Well it turns out to be: $\sqrt[r]{\frac{1}{b}}$.

Let's give a quick example of b = 20 bands of r = 5 rows per band. Then the function: $1 - (1 - s^r)^b$ takes the form of: $1 - (1 - s^5)^{20}$. In Figure 5.4 below we give the result of the function for some values of the true similarity s. What we notice is that the biggest leap the function takes is from s = 0.4 to s = 0.6, it rises more than 0.6.

Figure 5.4 : Values of the S-Curve for b = 20 and r = 5 [1]

s	$1-(1-s^r)^b$
.2	.006
.3	.047
.4	.186
.5	.470
.6	.802
.7	.975
.8	.9996

The figure above is essentially the probability that two columns will hash to the same bucket in at least one Band and essentialy these two columns become a candidate pair. The above figure is for b = 20 and r =5. Let’s make an example to see this better. Let’s suppose that for 2 columns in the characteristic Matrix we have true Jaccard Similarity s = 0.6, with the above b and r. If we do the math we have $(1-0.6^5)$ = 1 - 0.07776 = 0.92224.

0.92224^{20} = 0.19809754616

1 - 0.19809754616 = 0.80190245384

Again that is: $1-\ (1-0.6^5)^{20}$ = 0.80190245384

So what this means is that in one band the probability for all rows to agree is

0.6^5= 7.77%. But the probability there is at least one band from the 20 Bands that agrees in all rows is: 0.80190245384. So there is a possibility of 20% in this case to miss these pairs as false negatives.

5.4 Merging the Techniques

We can summarize now to the big picture of the Min-Hashing algorithm with the LSH technique and we can conclude to the construction of the algorithm in the following steps. But before we conclude to a concrete number of steps let’s recap some basic and fundamental concepts of the Min – Hash algorithm. It should be noticed that this technique produces false positives. That is pairs that marked as candidate pairs – hashed to the same bucket by accident (we discard them after they are found not to be actually similar). Min-Hash also produces false negatives, that is pairs that are truly similar but never found to be (never hashed in the same bucket in any of the Bucket Array Lists). This case is what we are trying more to avoid. Based on the discussion in the previous chapters, we finally provide the steps for the algorithm.

Ask user for the number k that denotes the length of k – shingles. Pick the right k number depending on how big the sets would be. For example k = 5 for e-mails, k = 9 or 10 for bigger documents of 10 pages or more long, e.t.c. Construct from each document the k – shingle set and pass it through a good hash function (we use Mur-Mur Hash 3 in our case), so you are dealing with plain numbers instead of strings.

1) Short the characteristic Matrix by shingle to have the Universe of numbers – shingles.
2) Prompt user for the number of hash functions and construct the Signature Matrix from the algorithm 4.1 we gave in chapter 4.
3) Prompt user for a desired similarity threshold s. Tune the number of Bands b and number of rows r so that $b \times r = n$, and s ≈ $\sqrt[r]{\frac{1}{b}}$. Ask user if speed or accuracy is desired. If user selected accuracy (that means avoiding false negatives) then pick b and r in a way such that they give a value slightly lower that s in the above S – Curve function: $\sqrt[r]{\frac{1}{b}}$. Accuracy actually means more bands with lesser rows per band. Otherwise if user selected speed, then that value can be slightly bigger. Speed actually means to run the algorithm faster thereby avoiding false positives. That means lesser bands with more rows per band. It means that if two columns manage to hash to the same bucket in one bucket Array List, then these columns almost certainly are identical.
4) Apply the LSH technique as we explained in section 5.2 in the Signature Matrix.
5) Check if the candidate pairs found by the LSH pass the similarity threshold given by the user. That means firstly checking if the signatures of the candidate pairs found by the LSH are over the similarity threshold. If they are not, filter them out.
6) For those signature pairs that pass the similarity threshold, we calculate the true Jaccard Similarity from their characteristic Matrix, to see that they are truly similar.

The summarization of all the above steps and stages we followed to produce the final result; that is the final candidate pairs of documents, is depicted in the initial picture 3.1, that showed us the big picture of the MinHashing Algorithm.

6 Conclusions

In this thesis, we have explored similarity search problems using techniques such as Minhashing and Locality Sensitive Hashing. We saw in detail the inner workings of the Minhashing algorithm and how it preserves the similarity of 2 columns in the MinHash Signature Matrix while greatly shrinking the size of every document. We scrutinized the direct correlation MinHash signatures and the signature matrix have with the Jaccard Similarity.

We also examined the Locality Sensitive Hashing, through the Banding technique which was based upon the MinHash Signature Matrix. We hashed the Signature Matrix to a big number of buckets, so that documents that hashed to the same buckets, formed candidate pairs. We also saw the probability of a "good" estimation in finding the candidate pairs, with regard to the right pair of Rows r and Bands b and a given similarity threshold.

6.1 Applications and Other Uses

Min – Hash algorithm is found to have many applications like clustering and near – duplicate elimination. Among many applications Min-Hash algorithm has been used for finding:

- Mirror websites, or approximate mirrors
- Plagiarism detection
- Spam identification (e-mail, etc.)
- Near duplicate documents, news articles, etc.
- Document clustering and other.

There are many other similar techniques that use the Min – Hash algorithm in other cases like image data processing.

As previously mentioned the Locality Sensitive Hashing technique is a dimensionality reduction form and has many cases and illustrations in many N – N problems. LSH can also use a plethora of techniques for estimating the distance like Haming, Cosine distance and others, which we are not going to describe here.

The Min-Hash algorithm may be seen as a special case of LSH that makes extensive use of hashing algorithms. In the more general Map-Reduce scheme, there are also many similar versions of the MinHash Algorithm that are used in more specific way, depending on the particular problem.

6.2 Evaluation of the Algorithm

Min-Hash and other algorithms have been tested and evaluated by big companies and corporations like Google. In 2007, Google highlighted the Min-Hash algorithm with LSH to be more appropriate for Google News personalization.

Bibliography

[1] Jure Leskovec, Anand Rajaraman, Jeff Ullman, “Stanford University” “Mining of Massive Datasets” Book [online] available at http://www.mmds.org/

[2] From the 1995 *Random House Compact Unabridged Dictionary*: use or close imitation of the language and thoughts of another author and the representation of them as one's own original work qtd. in *Stepchyshyn, Vera; Nelson, Robert S. (2007). Library plagiarism policies. Assoc. of College & Resrch Libraries. p. 65. ISBN 0-8389-8416-9.*

From the Oxford English Dictionary: the wrongful appropriation or purloining and publication as one's own, of the ideas, or the expression of the ideas… of another

qtd. in Lands (1999)

[3] Valpy, Francis Edward Jackson (2005) *Etymological Dictionary of the Latin Language*, p.345 entry for *plagium*, quotation: "the crime of kidnapping."

[4] *Broder, Andrei Z. (1997), "On the resemblance and containment of documents", Compression and Complexity of Sequences: Proceedings, Positano, Amalfitan Coast, Salerno, Italy, June 11-13, 1997(PDF),*

[5] *Zhao, Kang; Lu, Hongtao; Mei, Jincheng (2014).* *"Locality Preserving Hashing". pp. 2874–2880. Tsai, Yi-Hsuan; Yang, Ming-Hsuan (October 2014). "Locality preserving hashing". pp. 2988–2992.*

[6] https://www.princeton.edu/pr/pub/integrity/pages/plagiarism/

[7] https://en.wikipedia.org/wiki/Jaccard_index

7 Appendix

Below we will show the data samples it was tested, the source code and some results. Data samples were derived from a well-known excerpt of a famous poet. [6]

Let’s begin with the data samples we tested it. Please note that due to insufficient access to a cluster, i created and tested the source code in my personal laptop.

This program is written in Apache Spark which is an open-source cluster-computing framework (original author Matei Zaharia), and in Scala programming language, which is a functional language. Apache Spark is used for programming entire clusters with respect to data parallelism and fault tolerance.

So it can be used and run in a cluster-computing framework, (maybe with minor modifications). I created the source code with parallel techniques of programming in mind from the very first place (so that it can run on a cluster mode).

Platform and OS used:

- Ubuntu 16.04, IntelijIDE,
- Spark, Scala (You can see versions from the .sbt file)

Below I give some data samples that i tested it on.

7.1 Data Samples

Below are given some examples of our test Set that vary from verbatim plagiarism, to paraphrasing.

Haml_Othello_Original.txt

From time to time this submerged or latent theater in Hamlet becomes almost overt. It is close to the surface in Hamlet's pretense of madness, the "antic disposition" he puts on to protect himself and prevent his antagonists from plucking out the heart of his mystery. It is even closer to the surface when Hamlet enters his mother's room and holds up, side by side, the pictures of the two kings, Old Hamlet and Claudius, and proceeds to describe for her the true nature of the choice she has made, presenting truth by means of a show. Similarly, when he leaps into the open grave at Ophelia's funeral, ranting in high heroic terms, he is acting out for Laertes, and perhaps for himself as well, the folly of excessive, melodramatic expressions of grief.

Haml_Oth_1.txt (Verbatim plagiarism, or unacknowledged direct quotation (lifted passages are underlined))

Almost all of Shakespeare's *Hamlet* can be understood as a play about acting and the theater. For example, there is Hamlet's pretense of madness, the "antic disposition" that he puts on to protect himself and prevent his antagonists from plucking out the heart of his mystery. When Hamlet enters his mother's room, he holds up, side by side, the pictures of the two kings, Old Hamlet and Claudius, and proceeds to describe for her the true nature of the choice she has made, presenting truth by means of a show. Similarly, when he leaps into the open grave at Ophelia's funeral, ranting in high heroic terms, he is acting out for Laertes, and perhaps for himself as well, the folly of excessive, melodramatic expressions of grief.

Haml_Oth_2.txt (Lifting selected passages and phrases without proper acknowledgment (lifted passages are underlined))

Almost all of Shakespeare's *Hamlet* can be understood as a play about acting and the theater. For example, in Act 1, Hamlet adopts a pretense of madness that he uses to protect himself and prevent his antagonists from discovering his mission to revenge his father's murder. He also presents truth by means of a show when he compares the portraits of Gertrude's two husbands in order to describe for her the true nature of the choice she has made. And when he leaps in Ophelia’s open grave ranting in high heroic terms, Hamlet is acting out the folly of excessive, melodramatic expressions of grief.

Haml_Oth_3.txt (Paraphrasing the text while maintaining the basic paragraph and sentence structure)

Almost all of Shakespeare’s *Hamlet* can be understood as a play about acting and the theater. For example, in Act 1, Hamlet pretends to be insane in order to make sure his enemies do not discover his mission to revenge his father’s murder. The theme is even more obvious when Hamlet compares the pictures of his mother’s two husbands to show her what a bad choice she has made, using their images to reveal the truth. Also, when he jumps into Ophelia’s grave, hurling his challenge to Laertes, Hamlet demonstrates the foolishness of exaggerated expressions of emotion.

Note that highlighted texts: **Haml_Oth_1.txt** and **Haml_Othello_Original.txt** which are verbatim plagiarism, are more likely to be found plagiarized from the MinHash algorithm because they have the more shingles in common than the others (see the underlined text).

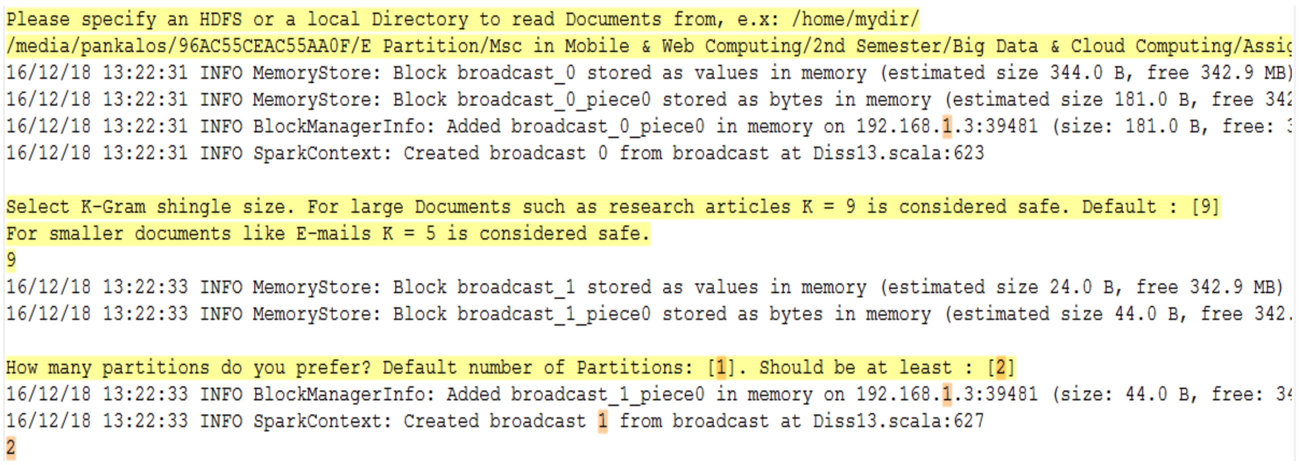
Following are some results from the program:

7.2 Some Results

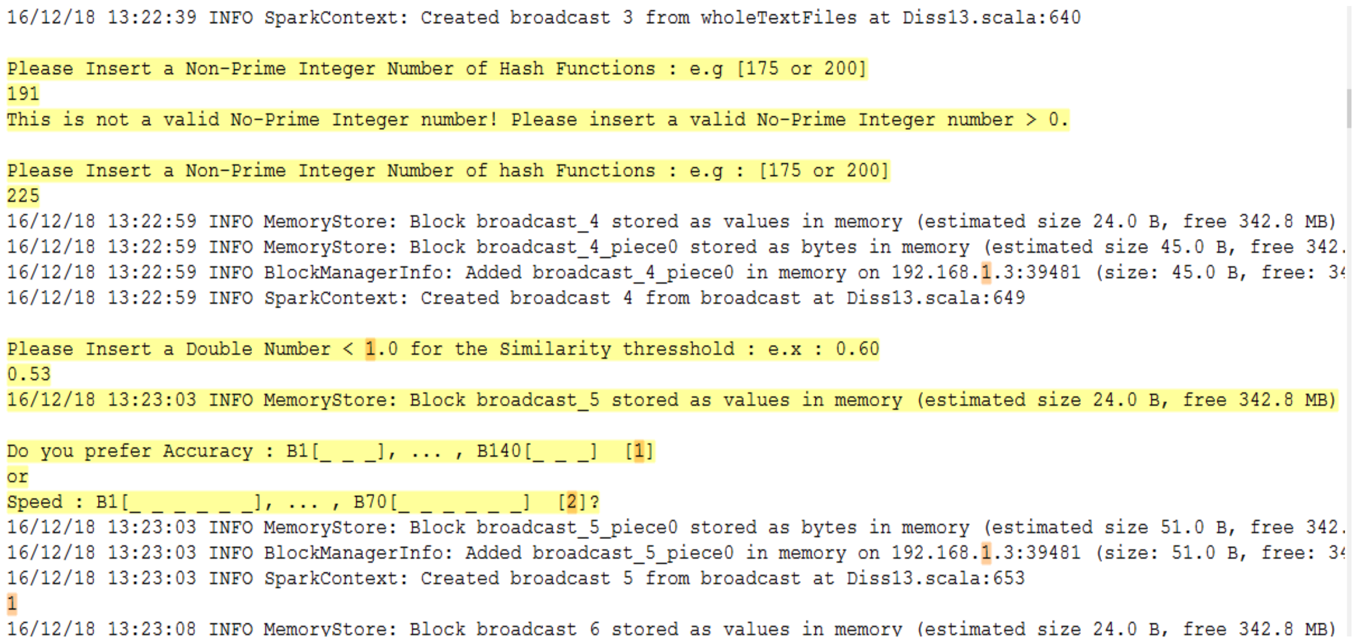
First of all the program prompts the user for some essential input variables like:

- The path Directory where all the files exist.
- N-Gram size of shingles
- The level of parallelism. That is how many partitions user prefers. (Should be at least 2 to see a level of parallelism). The default should be at least the number of cores in the whole cluster.
- The Number of Hash-Functions.
- The Similarity threshold above which the user prefers the documents to be similar.

- And a last parameter which denotes whether the user prefers Accuracy (more smaller bands but avoiding of false negatives) instead of Speed (Bigger bands, but bigger Risk of false negatives, but *Faster*).

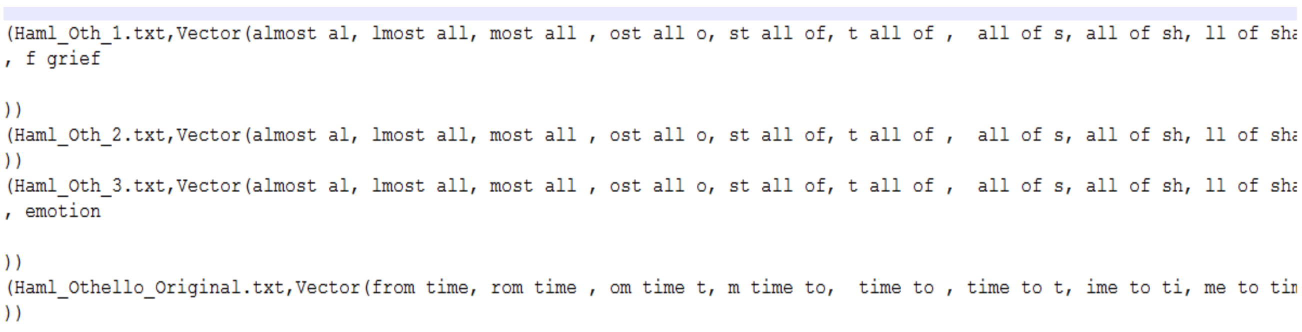


```
Please specify an HDFS or a local Directory to read Documents from, e.x: /home/mydir/
/media/pankalos/96AC55CEAC55AA0F/E Partition/Msc in Mobile & Web Computing/2nd Semester/Big Data & Cloud Computing/Assi
16/12/18 13:22:31 INFO MemoryStore: Block broadcast_0 stored as values in memory (estimated size 344.0 B, free 342.9 MB)
16/12/18 13:22:31 INFO MemoryStore: Block broadcast_0_piece0 stored as bytes in memory (estimated size 181.0 B, free 34
16/12/18 13:22:31 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on 192.168.1.3:39481 (size: 181.0 B, free:
16/12/18 13:22:31 INFO SparkContext: Created broadcast 0 from broadcast at Diss13.scala:623

Select K-Gram shingle size. For large Documents such as research articles K = 9 is considered safe. Default : [9]
For smaller documents like E-mails K = 5 is considered safe.
9
16/12/18 13:22:33 INFO MemoryStore: Block broadcast_1 stored as values in memory (estimated size 24.0 B, free 342.9 MB)
16/12/18 13:22:33 INFO MemoryStore: Block broadcast_1_piece0 stored as bytes in memory (estimated size 44.0 B, free 342.

How many partitions do you prefer? Default number of Partitions: [1]. Should be at least : [2]
16/12/18 13:22:33 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on 192.168.1.3:39481 (size: 44.0 B, free: 3
16/12/18 13:22:33 INFO SparkContext: Created broadcast 1 from broadcast at Diss13.scala:627
2
```



```
16/12/18 13:22:39 INFO SparkContext: Created broadcast 3 from wholeTextFiles at Diss13.scala:640

Please Insert a Non-Prime Integer Number of Hash Functions : e.g [175 or 200]
191
This is not a valid No-Prime Integer number! Please insert a valid No-Prime Integer number > 0.

Please Insert a Non-Prime Integer Number of hash Functions : e.g : [175 or 200]
225
16/12/18 13:22:59 INFO MemoryStore: Block broadcast_4 stored as values in memory (estimated size 24.0 B, free 342.8 MB)
16/12/18 13:22:59 INFO MemoryStore: Block broadcast_4_piece0 stored as bytes in memory (estimated size 45.0 B, free 342.
16/12/18 13:22:59 INFO BlockManagerInfo: Added broadcast_4_piece0 in memory on 192.168.1.3:39481 (size: 45.0 B, free: 3
16/12/18 13:22:59 INFO SparkContext: Created broadcast 4 from broadcast at Diss13.scala:649

Please Insert a Double Number < 1.0 for the Similarity thresshold : e.x : 0.60
0.53
16/12/18 13:23:03 INFO MemoryStore: Block broadcast_5 stored as values in memory (estimated size 24.0 B, free 342.8 MB)

Do you prefer Accuracy : B1[_ _ _], ... , B140[_ _ _]  [1]
or
Speed : B1[_ _ _ _ _ _], ... , B70[_ _ _ _ _ _]  [2]?
16/12/18 13:23:03 INFO MemoryStore: Block broadcast_5_piece0 stored as bytes in memory (estimated size 51.0 B, free 342.
16/12/18 13:23:03 INFO BlockManagerInfo: Added broadcast_5_piece0 in memory on 192.168.1.3:39481 (size: 51.0 B, free: 3
16/12/18 13:23:03 INFO SparkContext: Created broadcast 5 from broadcast at Diss13.scala:653
1
16/12/18 13:23:08 INFO MemoryStore: Block broadcast 6 stored as values in memory (estimated size 24.0 B, free 342.8 MB)
```

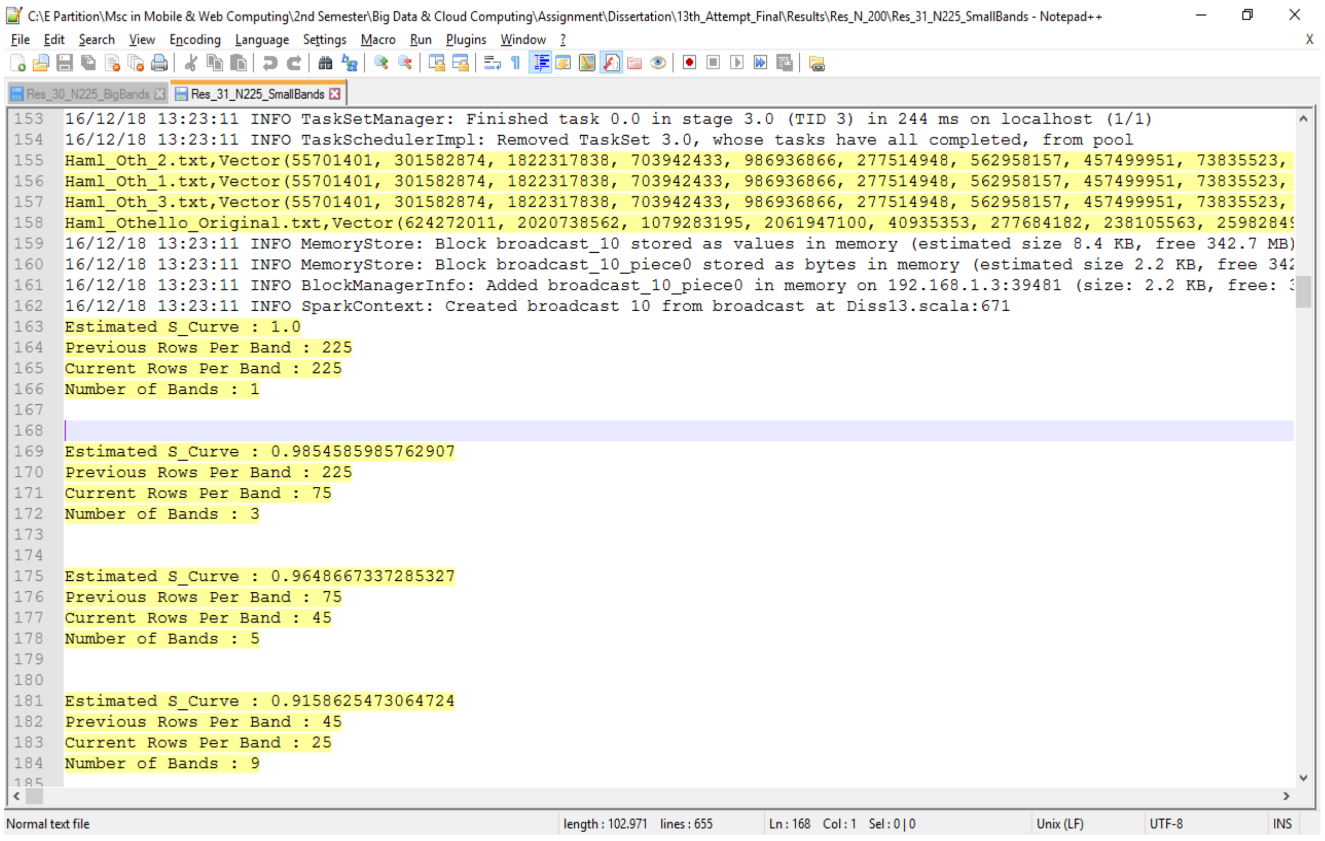
7.2.1 The Shingling phase



```
(Haml_Oth_1.txt,Vector(almost al, lmost all, most all , ost all o, st all of, t all of ,  all of s, all of sh, ll of sh
, f grief

))
(Haml_Oth_2.txt,Vector(almost al, lmost all, most all , ost all o, st all of, t all of ,  all of s, all of sh, ll of sh
))
(Haml_Oth_3.txt,Vector(almost al, lmost all, most all , ost all o, st all of, t all of ,  all of s, all of sh, ll of sh
, emotion

))
(Haml_Othello_Original.txt,Vector(from time, rom time , om time t, m time to,  time to , time to t, ime to ti, me to ti
))
```

That is the shingling phase: We extract n-shingles from the original document.

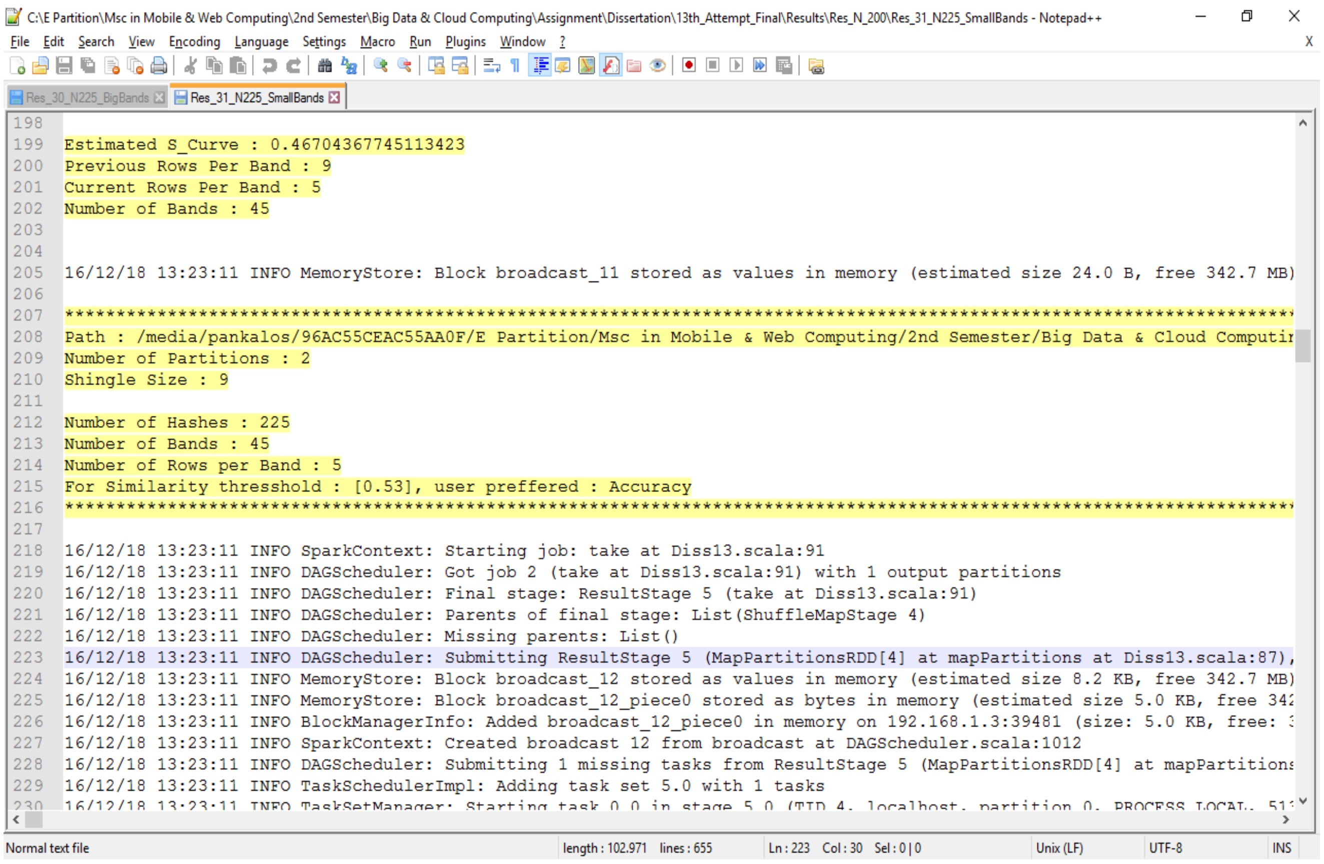
7.2.2 Hashing the Shingles


C:\E Partition\Msc in Mobile & Web Computing\2nd Semester\Big Data & Cloud Computing\Assignment\Dissertation\13th_Attempt_Final\Results\Res_N_200\Res_31_N225_SmallBands - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Res_30_N225_BigBands
Res_31_N225_SmallBands
Normal text file
length : 102.971 lines : 655
Ln : 168 Col : 1 Sel : 0 | 0
Unix (LF)
UTF-8
INS

```
16/12/18 13:23:11 INFO TaskSetManager: Finished task 0.0 in stage 3.0 (TID 3) in 244 ms on localhost (1/1)
16/12/18 13:23:11 INFO TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have all completed, from pool
Haml_Oth_2.txt,Vector(55701401, 301582874, 1822317838, 703942433, 986936866, 277514948, 562958157, 457499951, 73835523,
Haml_Oth_1.txt,Vector(55701401, 301582874, 1822317838, 703942433, 986936866, 277514948, 562958157, 457499951, 73835523,
Haml_Oth_3.txt,Vector(55701401, 301582874, 1822317838, 703942433, 986936866, 277514948, 562958157, 457499951, 73835523,
Haml_Othello_Original.txt,Vector(624272011, 2020738562, 1079283195, 2061947100, 40935353, 277684182, 238105563, 2598284
16/12/18 13:23:11 INFO MemoryStore: Block broadcast_10 stored as values in memory (estimated size 8.4 KB, free 342.7 MB)
16/12/18 13:23:11 INFO MemoryStore: Block broadcast_10_piece0 stored as bytes in memory (estimated size 2.2 KB, free 342
16/12/18 13:23:11 INFO BlockManagerInfo: Added broadcast_10_piece0 in memory on 192.168.1.3:39481 (size: 2.2 KB, free: 
16/12/18 13:23:11 INFO SparkContext: Created broadcast 10 from broadcast at Diss13.scala:671
Estimated S_Curve : 1.0
Previous Rows Per Band : 225
Current Rows Per Band : 225
Number of Bands : 1


Estimated S_Curve : 0.9854585985762907
Previous Rows Per Band : 225
Current Rows Per Band : 75
Number of Bands : 3


Estimated S_Curve : 0.9648667337285327
Previous Rows Per Band : 75
Current Rows Per Band : 45
Number of Bands : 5


Estimated S_Curve : 0.9158625473064724
Previous Rows Per Band : 45
Current Rows Per Band : 25
Number of Bands : 9
```

From the above picture we can see that we have hashed the Document shingles to Integers. So we have Vector[Int] now to play with.

It is also shown how the tuning process of the number of bands and rows per band takes place, until we converge to the similarity threshold.

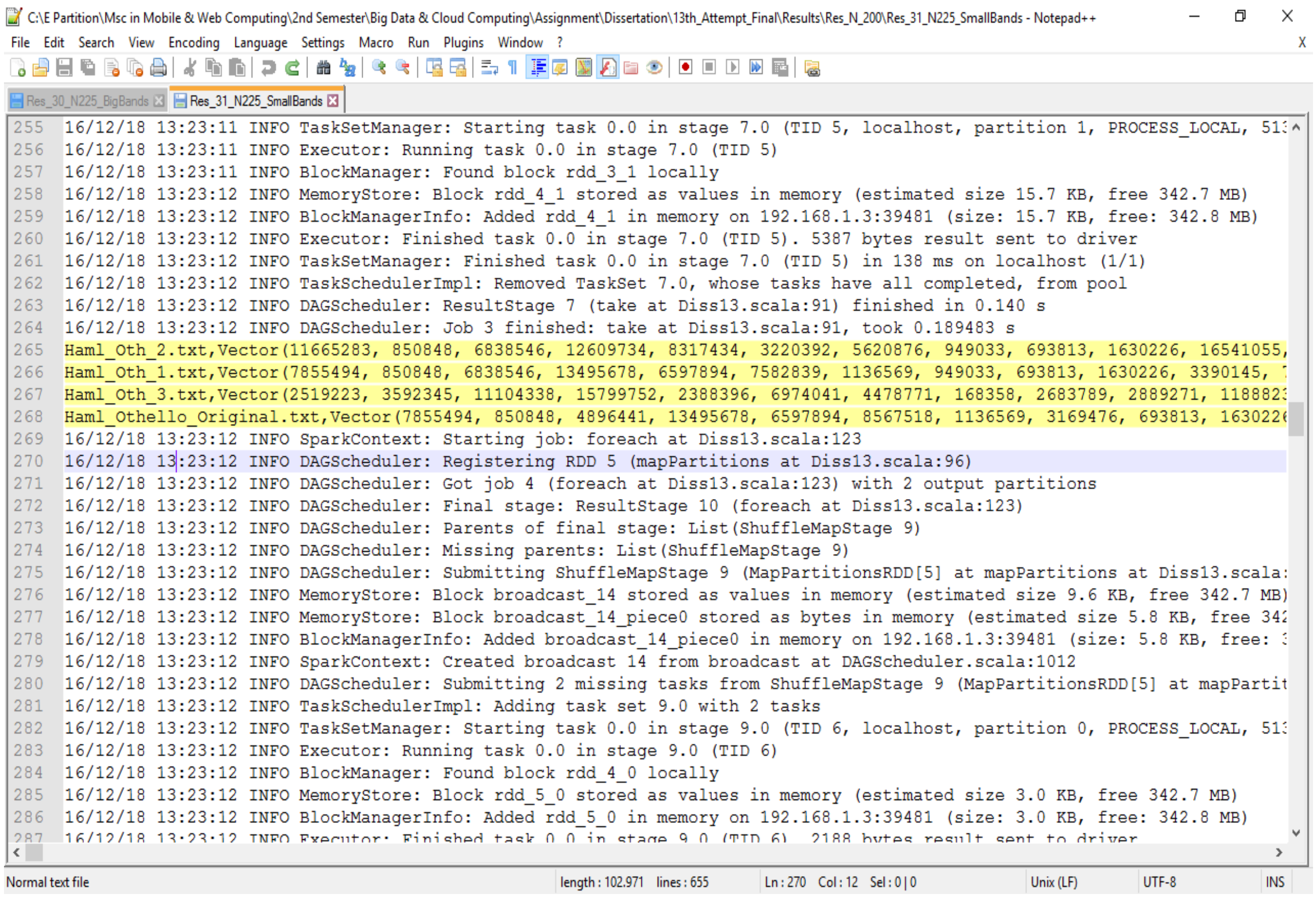

C:\E Partition\Msc in Mobile & Web Computing\2nd Semester\Big Data & Cloud Computing\Assignment\Dissertation\13th_Attempt_Final\Results\Res_N_200\Res_31_N225_SmallBands - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Res_30_N225_BigBands
Res_31_N225_SmallBands
Normal text file
length : 102.971 lines : 655
Ln : 223 Col : 30 Sel : 0 | 0
Unix (LF)
UTF-8
INS

```

Estimated S_Curve : 0.46704367745113423
Previous Rows Per Band : 9
Current Rows Per Band : 5
Number of Bands : 45


16/12/18 13:23:11 INFO MemoryStore: Block broadcast_11 stored as values in memory (estimated size 24.0 B, free 342.7 MB)

************************************************************************************************************************
Path : /media/pankalos/96AC55CEAC55AA0F/E Partition/Msc in Mobile & Web Computing/2nd Semester/Big Data & Cloud Computi
Number of Partitions : 2
Shingle Size : 9

Number of Hashes : 225
Number of Bands : 45
Number of Rows per Band : 5
For Similarity thresshold : [0.53], user preffered : Accuracy
************************************************************************************************************************

16/12/18 13:23:11 INFO SparkContext: Starting job: take at Diss13.scala:91
16/12/18 13:23:11 INFO DAGScheduler: Got job 2 (take at Diss13.scala:91) with 1 output partitions
16/12/18 13:23:11 INFO DAGScheduler: Final stage: ResultStage 5 (take at Diss13.scala:91)
16/12/18 13:23:11 INFO DAGScheduler: Parents of final stage: List(ShuffleMapStage 4)
16/12/18 13:23:11 INFO DAGScheduler: Missing parents: List()
16/12/18 13:23:11 INFO DAGScheduler: Submitting ResultStage 5 (MapPartitionsRDD[4] at mapPartitions at Diss13.scala:87),
16/12/18 13:23:11 INFO MemoryStore: Block broadcast_12 stored as values in memory (estimated size 8.2 KB, free 342.7 MB)
16/12/18 13:23:11 INFO MemoryStore: Block broadcast_12_piece0 stored as bytes in memory (estimated size 5.0 KB, free 342
16/12/18 13:23:11 INFO BlockManagerInfo: Added broadcast_12_piece0 in memory on 192.168.1.3:39481 (size: 5.0 KB, free: 
16/12/18 13:23:11 INFO SparkContext: Created broadcast 12 from broadcast at DAGScheduler.scala:1012
16/12/18 13:23:11 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 5 (MapPartitionsRDD[4] at mapPartition
16/12/18 13:23:11 INFO TaskSchedulerImpl: Adding task set 5.0 with 1 tasks
16/12/18 13:23:11 INFO TaskSetManager: Starting task 0.0 in stage 5.0 (TID 4, localhost, partition 0, PROCESS_LOCAL, 51
```

The above picture shows all the final parameters that were calculated and/or selected by the user.

7.2.3 The MinHash Signature

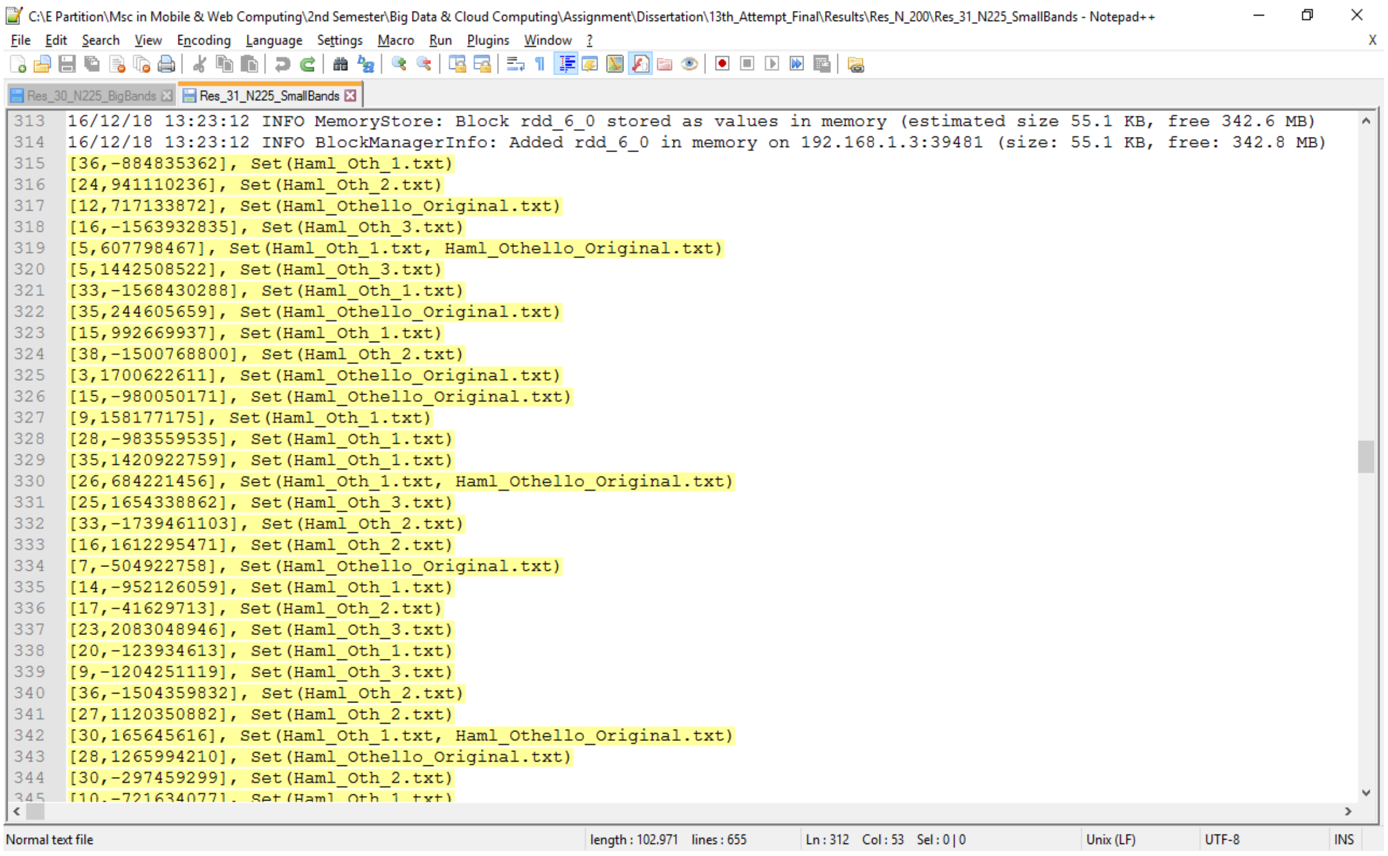
Below are the MinHash Signatures of every document. Note that all Vectors contain N Hash Integers (the User input, in our case is 225).


C:\E Partition\Msc in Mobile & Web Computing\2nd Semester\Big Data & Cloud Computing\Assignment\Dissertation\13th_Attempt_Final\Results\Res_N_200\Res_31_N225_SmallBands - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Res_30_N225_BigBands
Res_31_N225_SmallBands
Normal text file
length : 102.971 lines : 655
Ln : 270 Col : 12 Sel : 0 | 0
Unix (LF)
UTF-8
INS

```
16/12/18 13:23:11 INFO TaskSetManager: Starting task 0.0 in stage 7.0 (TID 5, localhost, partition 1, PROCESS_LOCAL, 51
16/12/18 13:23:11 INFO Executor: Running task 0.0 in stage 7.0 (TID 5)
16/12/18 13:23:11 INFO BlockManager: Found block rdd_3_1 locally
16/12/18 13:23:12 INFO MemoryStore: Block rdd_4_1 stored as values in memory (estimated size 15.7 KB, free 342.7 MB)
16/12/18 13:23:12 INFO BlockManagerInfo: Added rdd_4_1 in memory on 192.168.1.3:39481 (size: 15.7 KB, free: 342.8 MB)
16/12/18 13:23:12 INFO Executor: Finished task 0.0 in stage 7.0 (TID 5). 5387 bytes result sent to driver
16/12/18 13:23:12 INFO TaskSetManager: Finished task 0.0 in stage 7.0 (TID 5) in 138 ms on localhost (1/1)
16/12/18 13:23:12 INFO TaskSchedulerImpl: Removed TaskSet 7.0, whose tasks have all completed, from pool
16/12/18 13:23:12 INFO DAGScheduler: ResultStage 7 (take at Diss13.scala:91) finished in 0.140 s
16/12/18 13:23:12 INFO DAGScheduler: Job 3 finished: take at Diss13.scala:91, took 0.189483 s
Haml_Oth_2.txt,Vector(11665283, 850848, 6838546, 12609734, 8317434, 3220392, 5620876, 949033, 693813, 1630226, 16541055,
Haml_Oth_1.txt,Vector(7855494, 850848, 6838546, 13495678, 6597894, 7582839, 1136569, 949033, 693813, 1630226, 3390145,
Haml_Oth_3.txt,Vector(2519223, 3592345, 11104338, 15799752, 2388396, 6974041, 4478771, 168358, 2683789, 2889271, 118882
Haml_Othello_Original.txt,Vector(7855494, 850848, 4896441, 13495678, 6597894, 8567518, 1136569, 3169476, 693813, 163022
16/12/18 13:23:12 INFO SparkContext: Starting job: foreach at Diss13.scala:123
16/12/18 13:23:12 INFO DAGScheduler: Registering RDD 5 (mapPartitions at Diss13.scala:96)
16/12/18 13:23:12 INFO DAGScheduler: Got job 4 (foreach at Diss13.scala:123) with 2 output partitions
16/12/18 13:23:12 INFO DAGScheduler: Final stage: ResultStage 10 (foreach at Diss13.scala:123)
16/12/18 13:23:12 INFO DAGScheduler: Parents of final stage: List(ShuffleMapStage 9)
16/12/18 13:23:12 INFO DAGScheduler: Missing parents: List(ShuffleMapStage 9)
16/12/18 13:23:12 INFO DAGScheduler: Submitting ShuffleMapStage 9 (MapPartitionsRDD[5] at mapPartitions at Diss13.scala
16/12/18 13:23:12 INFO MemoryStore: Block broadcast_14 stored as values in memory (estimated size 9.6 KB, free 342.7 MB)
16/12/18 13:23:12 INFO MemoryStore: Block broadcast_14_piece0 stored as bytes in memory (estimated size 5.8 KB, free 342
16/12/18 13:23:12 INFO BlockManagerInfo: Added broadcast_14_piece0 in memory on 192.168.1.3:39481 (size: 5.8 KB, free:
16/12/18 13:23:12 INFO SparkContext: Created broadcast 14 from broadcast at DAGScheduler.scala:1012
16/12/18 13:23:12 INFO DAGScheduler: Submitting 2 missing tasks from ShuffleMapStage 9 (MapPartitionsRDD[5] at mapPartit
16/12/18 13:23:12 INFO TaskSchedulerImpl: Adding task set 9.0 with 2 tasks
16/12/18 13:23:12 INFO TaskSetManager: Starting task 0.0 in stage 9.0 (TID 6, localhost, partition 0, PROCESS_LOCAL, 51
16/12/18 13:23:12 INFO Executor: Running task 0.0 in stage 9.0 (TID 6)
16/12/18 13:23:12 INFO BlockManager: Found block rdd_4_0 locally
16/12/18 13:23:12 INFO MemoryStore: Block rdd_5_0 stored as values in memory (estimated size 3.0 KB, free 342.7 MB)
16/12/18 13:23:12 INFO BlockManagerInfo: Added rdd_5_0 in memory on 192.168.1.3:39481 (size: 3.0 KB, free: 342.8 MB)
16/12/18 13:23:12 INFO Executor: Finished task 0.0 in stage 9.0 (TID 6). 2188 bytes result sent to driver
```

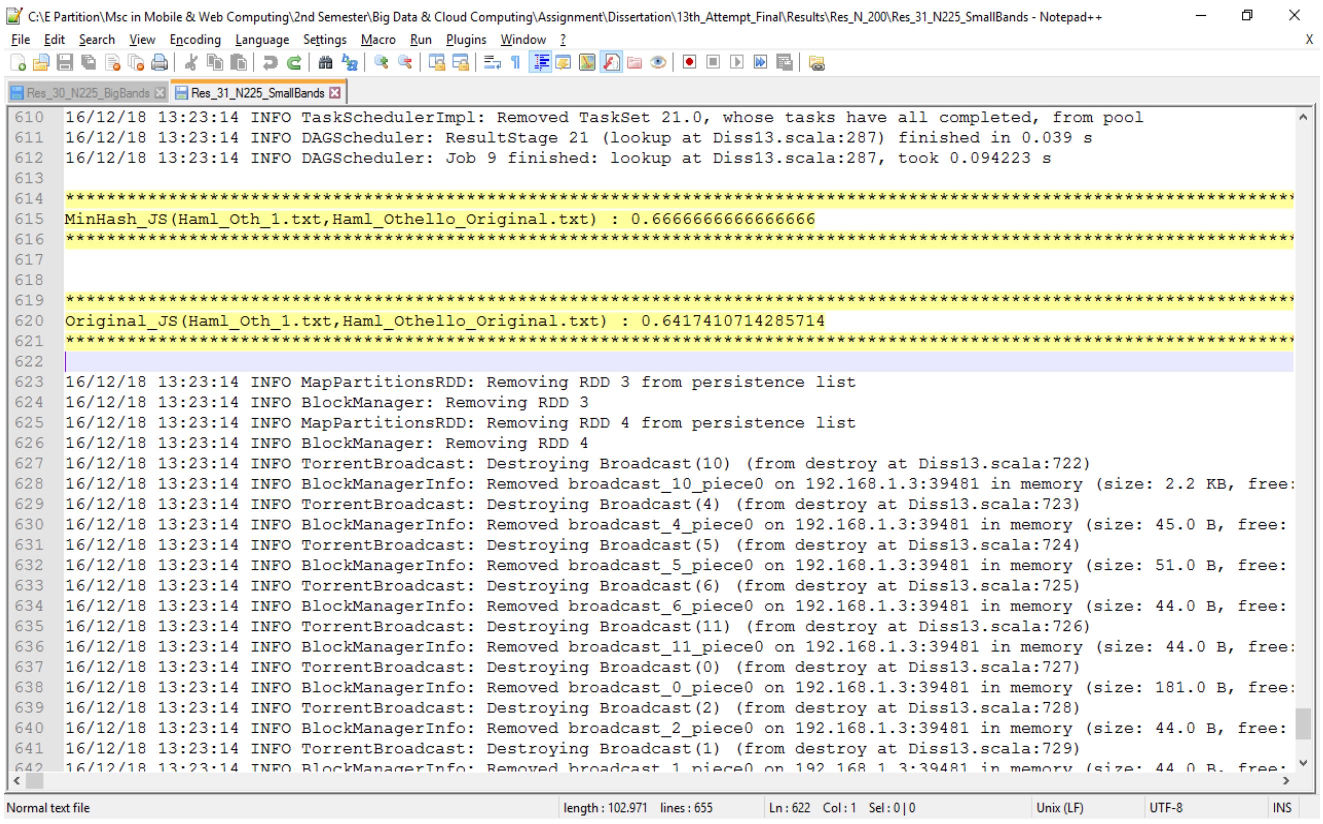
7.2.4 The LSH Banding Technique

The following picture depicts: How the MinHash Signature is hashed into a big number of buckets $\pm 2^{31} - 1$ in our case (that is $2^{32} - 1$ available buckets for every Band) and aggregated by (Band, Bucket). So those documents that hashed to the same bucket in some band, are a candidate pair. That is the LSH.


C:\E Partition\Msc in Mobile & Web Computing\2nd Semester\Big Data & Cloud Computing\Assignment\Dissertation\13th_Attempt_Final\Results\Res_N_200\Res_31_N225_SmallBands - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Res_30_N225_BigBands
Res_31_N225_SmallBands
Normal text file
length : 102.971 lines : 655
Ln : 312 Col : 53 Sel : 0 | 0
Unix (LF)
UTF-8
INS

```
16/12/18 13:23:12 INFO MemoryStore: Block rdd_6_0 stored as values in memory (estimated size 55.1 KB, free 342.6 MB)
16/12/18 13:23:12 INFO BlockManagerInfo: Added rdd_6_0 in memory on 192.168.1.3:39481 (size: 55.1 KB, free: 342.8 MB)
[36,-884835362], Set(Haml_Oth_1.txt)
[24,941110236], Set(Haml_Oth_2.txt)
[12,717133872], Set(Haml_Othello_Original.txt)
[16,-1563932835], Set(Haml_Oth_3.txt)
[5,607798467], Set(Haml_Oth_1.txt, Haml_Othello_Original.txt)
[5,1442508522], Set(Haml_Oth_3.txt)
[33,-1568430288], Set(Haml_Oth_1.txt)
[35,244605659], Set(Haml_Othello_Original.txt)
[15,992669937], Set(Haml_Oth_1.txt)
[38,-1500768800], Set(Haml_Oth_2.txt)
[3,1700622611], Set(Haml_Othello_Original.txt)
[15,-980050171], Set(Haml_Othello_Original.txt)
[9,158177175], Set(Haml_Oth_1.txt)
[28,-983559535], Set(Haml_Oth_1.txt)
[35,1420922759], Set(Haml_Oth_1.txt)
[26,684221456], Set(Haml_Oth_1.txt, Haml_Othello_Original.txt)
[25,1654338862], Set(Haml_Oth_3.txt)
[33,-1739461103], Set(Haml_Oth_2.txt)
[16,1612295471], Set(Haml_Oth_2.txt)
[7,-504922758], Set(Haml_Othello_Original.txt)
[14,-952126059], Set(Haml_Oth_1.txt)
[17,-41629713], Set(Haml_Oth_2.txt)
[23,2083048946], Set(Haml_Oth_3.txt)
[20,-123934613], Set(Haml_Oth_1.txt)
[9,-1204251119], Set(Haml_Oth_3.txt)
[36,-1504359832], Set(Haml_Oth_2.txt)
[27,1120350882], Set(Haml_Oth_2.txt)
[30,165645616], Set(Haml_Oth_1.txt, Haml_Othello_Original.txt)
[28,1265994210], Set(Haml_Othello_Original.txt)
[30,-297459299], Set(Haml_Oth_2.txt)
[10,-721634077], Set(Haml_Oth_1.txt)
```

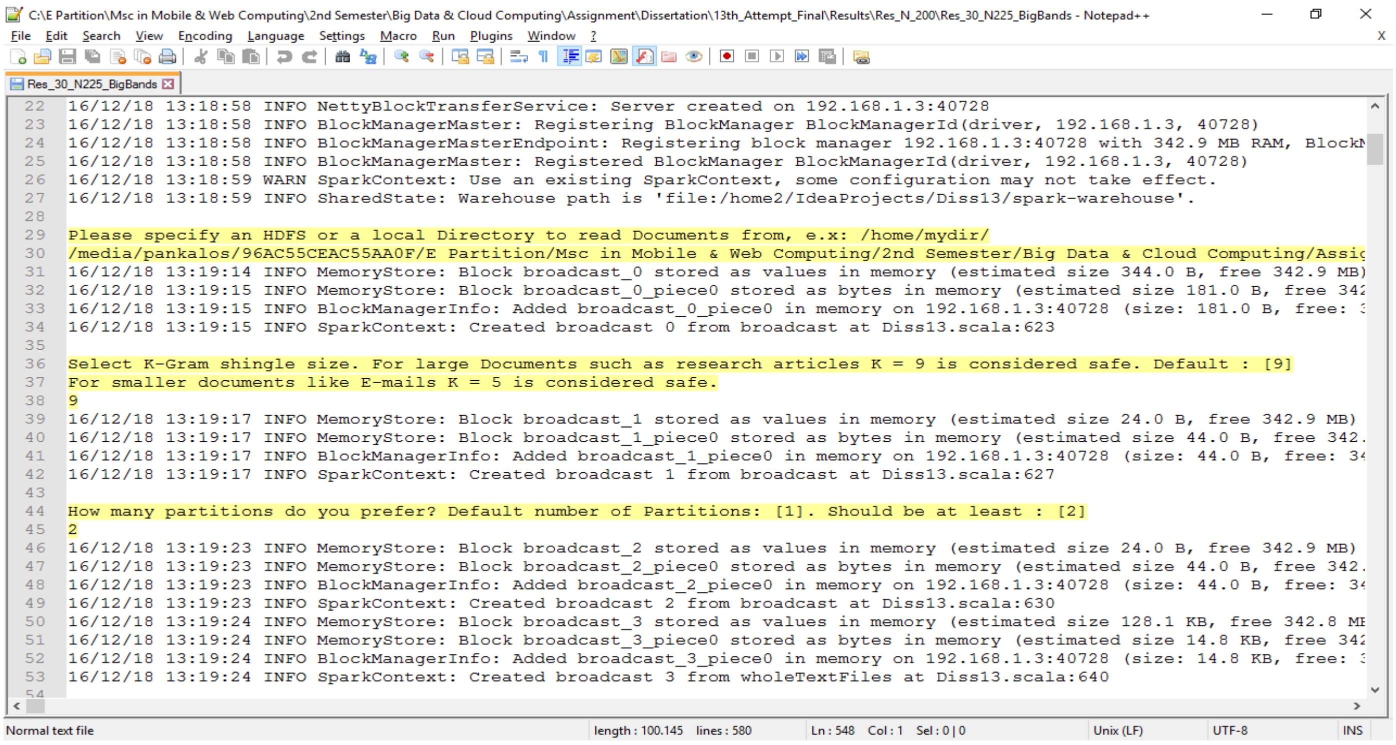
From the previous step take all the candidate pairs and find the Jaccard Similarity of their signatures, and the original Jaccard Similarity from the initial documents. (Documents that were hashed to the same bucket accidentally but the minHash Jaccard Similarity didn't exceed the similarity threshold were filtered out).


C:\E Partition\Msc in Mobile & Web Computing\2nd Semester\Big Data & Cloud Computing\Assignment\Dissertation\13th_Attempt_Final\Results\Res_N_200\Res_31_N225_SmallBands - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Res_30_N225_BigBands
Res_31_N225_SmallBands

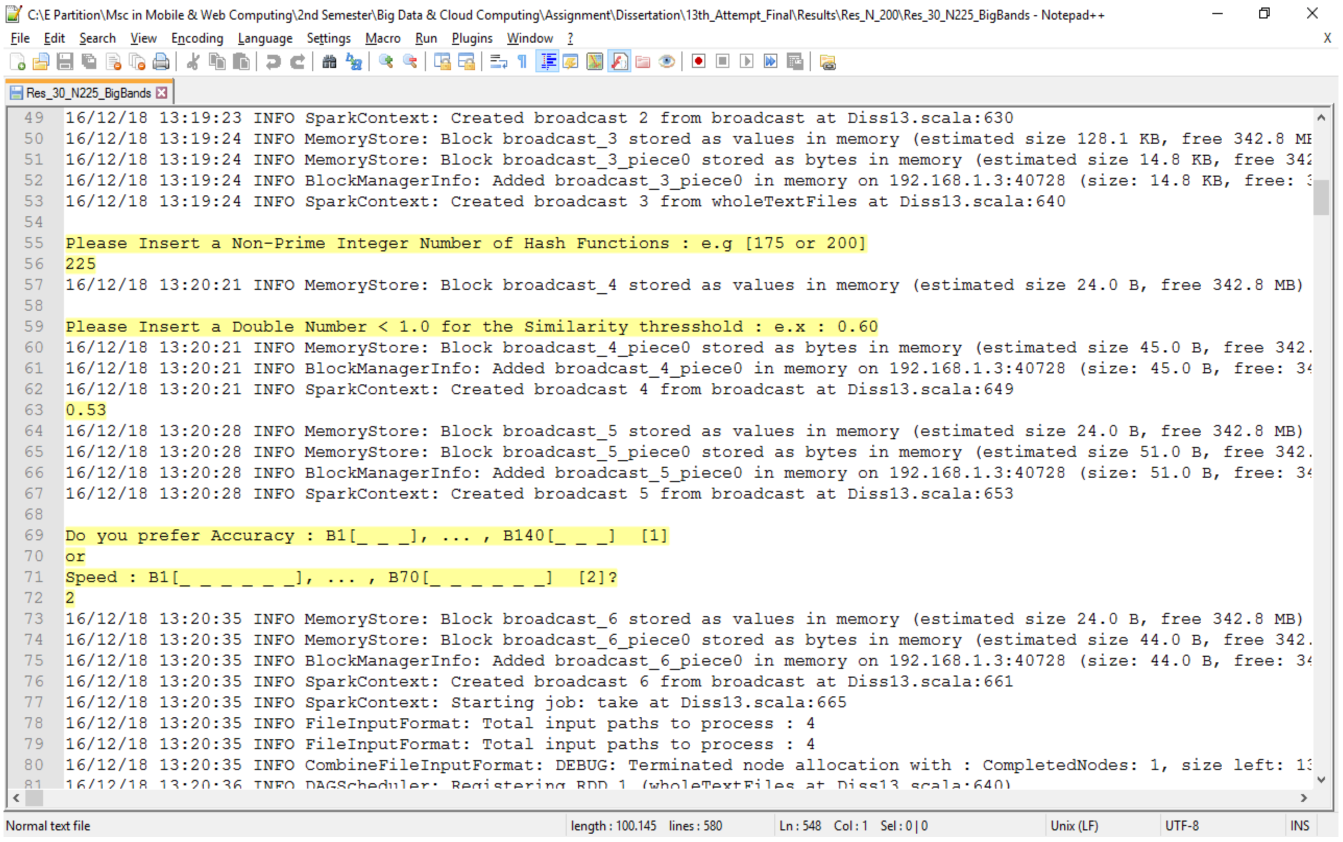
```
16/12/18 13:23:14 INFO TaskSchedulerImpl: Removed TaskSet 21.0, whose tasks have all completed, from pool
16/12/18 13:23:14 INFO DAGScheduler: ResultStage 21 (lookup at Diss13.scala:287) finished in 0.039 s
16/12/18 13:23:14 INFO DAGScheduler: Job 9 finished: lookup at Diss13.scala:287, took 0.094223 s

************************************************************************************************************************
MinHash_JS(Haml_Oth_1.txt,Haml_Othello_Original.txt) : 0.6666666666666666
************************************************************************************************************************


************************************************************************************************************************
Original_JS(Haml_Oth_1.txt,Haml_Othello_Original.txt) : 0.6417410714285714
************************************************************************************************************************

16/12/18 13:23:14 INFO MapPartitionsRDD: Removing RDD 3 from persistence list
16/12/18 13:23:14 INFO BlockManager: Removing RDD 3
16/12/18 13:23:14 INFO MapPartitionsRDD: Removing RDD 4 from persistence list
16/12/18 13:23:14 INFO BlockManager: Removing RDD 4
16/12/18 13:23:14 INFO TorrentBroadcast: Destroying Broadcast(10) (from destroy at Diss13.scala:722)
16/12/18 13:23:14 INFO BlockManagerInfo: Removed broadcast_10_piece0 on 192.168.1.3:39481 in memory (size: 2.2 KB, free:
16/12/18 13:23:14 INFO TorrentBroadcast: Destroying Broadcast(4) (from destroy at Diss13.scala:723)
16/12/18 13:23:14 INFO BlockManagerInfo: Removed broadcast_4_piece0 on 192.168.1.3:39481 in memory (size: 45.0 B, free:
16/12/18 13:23:14 INFO TorrentBroadcast: Destroying Broadcast(5) (from destroy at Diss13.scala:724)
16/12/18 13:23:14 INFO BlockManagerInfo: Removed broadcast_5_piece0 on 192.168.1.3:39481 in memory (size: 51.0 B, free:
16/12/18 13:23:14 INFO TorrentBroadcast: Destroying Broadcast(6) (from destroy at Diss13.scala:725)
16/12/18 13:23:14 INFO BlockManagerInfo: Removed broadcast_6_piece0 on 192.168.1.3:39481 in memory (size: 44.0 B, free:
16/12/18 13:23:14 INFO TorrentBroadcast: Destroying Broadcast(11) (from destroy at Diss13.scala:726)
16/12/18 13:23:14 INFO BlockManagerInfo: Removed broadcast_11_piece0 on 192.168.1.3:39481 in memory (size: 44.0 B, free:
16/12/18 13:23:14 INFO TorrentBroadcast: Destroying Broadcast(0) (from destroy at Diss13.scala:727)
16/12/18 13:23:14 INFO BlockManagerInfo: Removed broadcast_0_piece0 on 192.168.1.3:39481 in memory (size: 181.0 B, free:
16/12/18 13:23:14 INFO TorrentBroadcast: Destroying Broadcast(2) (from destroy at Diss13.scala:728)
16/12/18 13:23:14 INFO BlockManagerInfo: Removed broadcast_2_piece0 on 192.168.1.3:39481 in memory (size: 44.0 B, free:
16/12/18 13:23:14 INFO TorrentBroadcast: Destroying Broadcast(1) (from destroy at Diss13.scala:729)
16/12/18 13:23:14 INFO BlockManagerInfo: Removed broadcast_1_piece0 on 192.168.1.3:39481 in memory (size: 44.0 B, free:
```

Normal text file   length : 102.971   lines : 655   Ln : 622   Col : 1   Sel : 0 | 0   Unix (LF)   UTF-8   INS

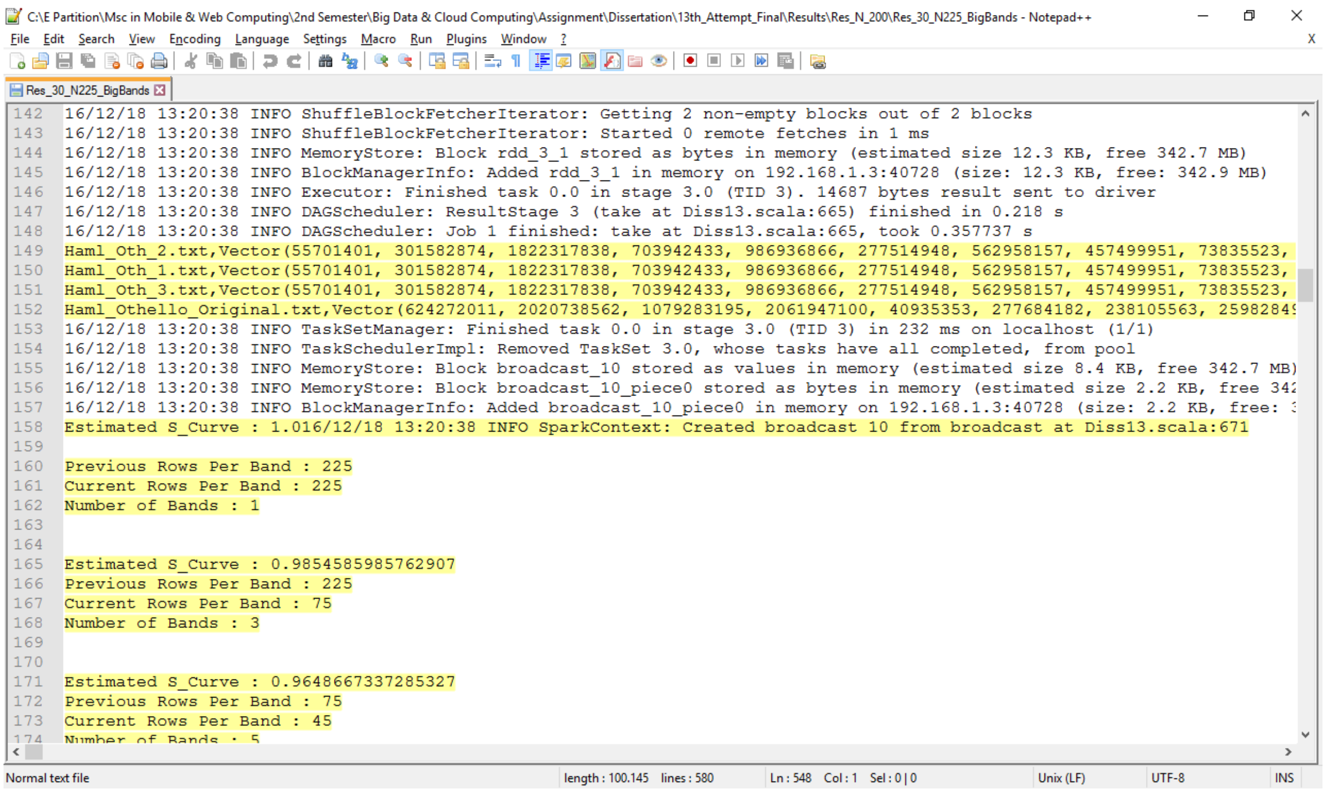
Let’s see the same example with “bigger” bands:



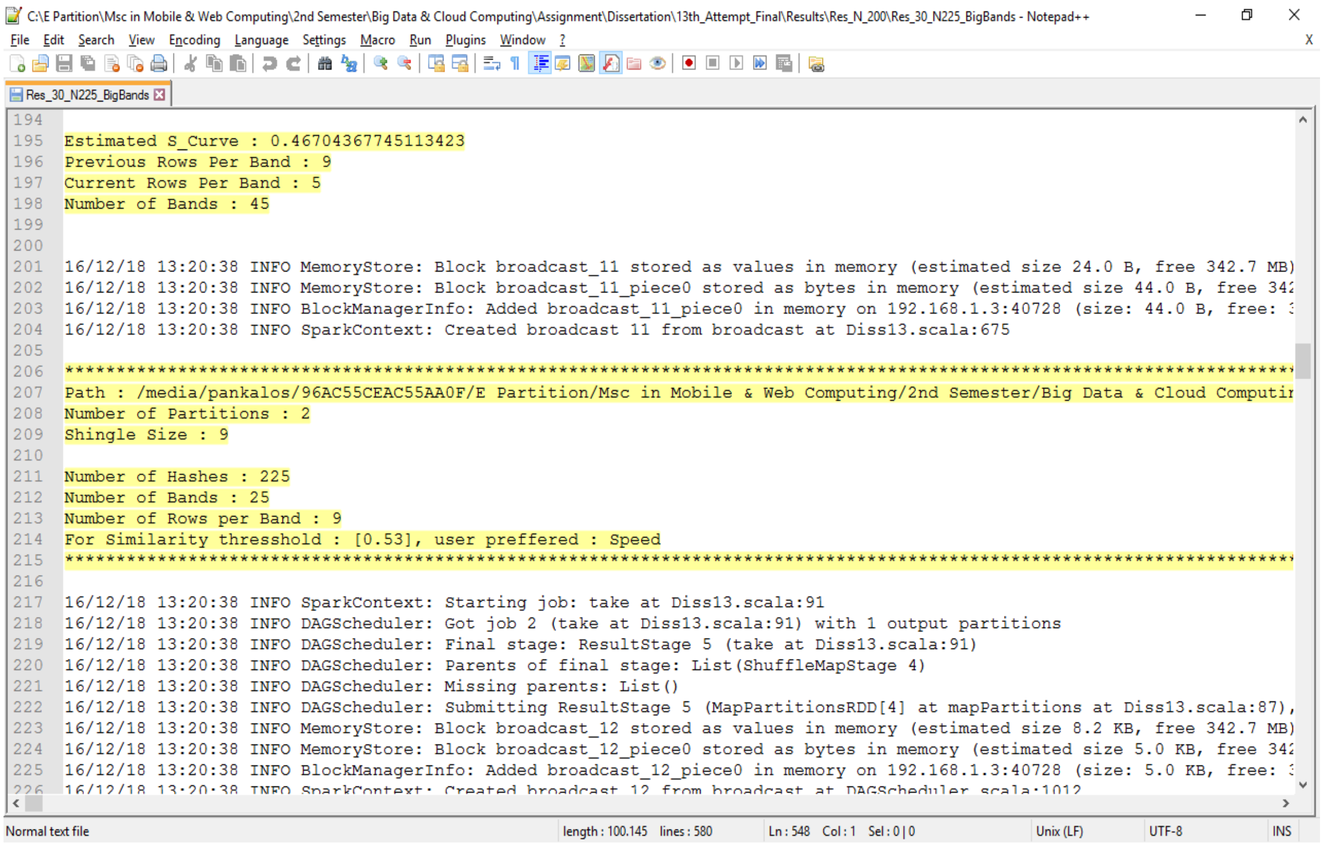
```
16/12/18 13:18:58 INFO NettyBlockTransferService: Server created on 192.168.1.3:40728
16/12/18 13:18:58 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, 192.168.1.3, 40728)
16/12/18 13:18:58 INFO BlockManagerMasterEndpoint: Registering block manager 192.168.1.3:40728 with 342.9 MB RAM, BlockM
16/12/18 13:18:58 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, 192.168.1.3, 40728)
16/12/18 13:18:59 WARN SparkContext: Use an existing SparkContext, some configuration may not take effect.
16/12/18 13:18:59 INFO SharedState: Warehouse path is 'file:/home2/IdeaProjects/Diss13/spark-warehouse'.

Please specify an HDFS or a local Directory to read Documents from, e.x: /home/mydir/
/media/pankalos/96AC55CEAC55AA0F/E Partition/Msc in Mobile & Web Computing/2nd Semester/Big Data & Cloud Computing/Assig
16/12/18 13:19:14 INFO MemoryStore: Block broadcast_0 stored as values in memory (estimated size 344.0 B, free 342.9 MB)
16/12/18 13:19:15 INFO MemoryStore: Block broadcast_0_piece0 stored as bytes in memory (estimated size 181.0 B, free 342
16/12/18 13:19:15 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on 192.168.1.3:40728 (size: 181.0 B, free: 3
16/12/18 13:19:15 INFO SparkContext: Created broadcast 0 from broadcast at Diss13.scala:623

Select K-Gram shingle size. For large Documents such as research articles K = 9 is considered safe. Default : [9]
For smaller documents like E-mails K = 5 is considered safe.
9
16/12/18 13:19:17 INFO MemoryStore: Block broadcast_1 stored as values in memory (estimated size 24.0 B, free 342.9 MB)
16/12/18 13:19:17 INFO MemoryStore: Block broadcast_1_piece0 stored as bytes in memory (estimated size 44.0 B, free 342.
16/12/18 13:19:17 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on 192.168.1.3:40728 (size: 44.0 B, free: 34
16/12/18 13:19:17 INFO SparkContext: Created broadcast 1 from broadcast at Diss13.scala:627

How many partitions do you prefer? Default number of Partitions: [1]. Should be at least : [2]
2
16/12/18 13:19:23 INFO MemoryStore: Block broadcast_2 stored as values in memory (estimated size 24.0 B, free 342.9 MB)
16/12/18 13:19:23 INFO MemoryStore: Block broadcast_2_piece0 stored as bytes in memory (estimated size 44.0 B, free 342.
16/12/18 13:19:23 INFO BlockManagerInfo: Added broadcast_2_piece0 in memory on 192.168.1.3:40728 (size: 44.0 B, free: 34
16/12/18 13:19:23 INFO SparkContext: Created broadcast 2 from broadcast at Diss13.scala:630
16/12/18 13:19:24 INFO MemoryStore: Block broadcast_3 stored as values in memory (estimated size 128.1 KB, free 342.8 MB
16/12/18 13:19:24 INFO MemoryStore: Block broadcast_3_piece0 stored as bytes in memory (estimated size 14.8 KB, free 342
16/12/18 13:19:24 INFO BlockManagerInfo: Added broadcast_3_piece0 in memory on 192.168.1.3:40728 (size: 14.8 KB, free: 3
16/12/18 13:19:24 INFO SparkContext: Created broadcast 3 from wholeTextFiles at Diss13.scala:640
```



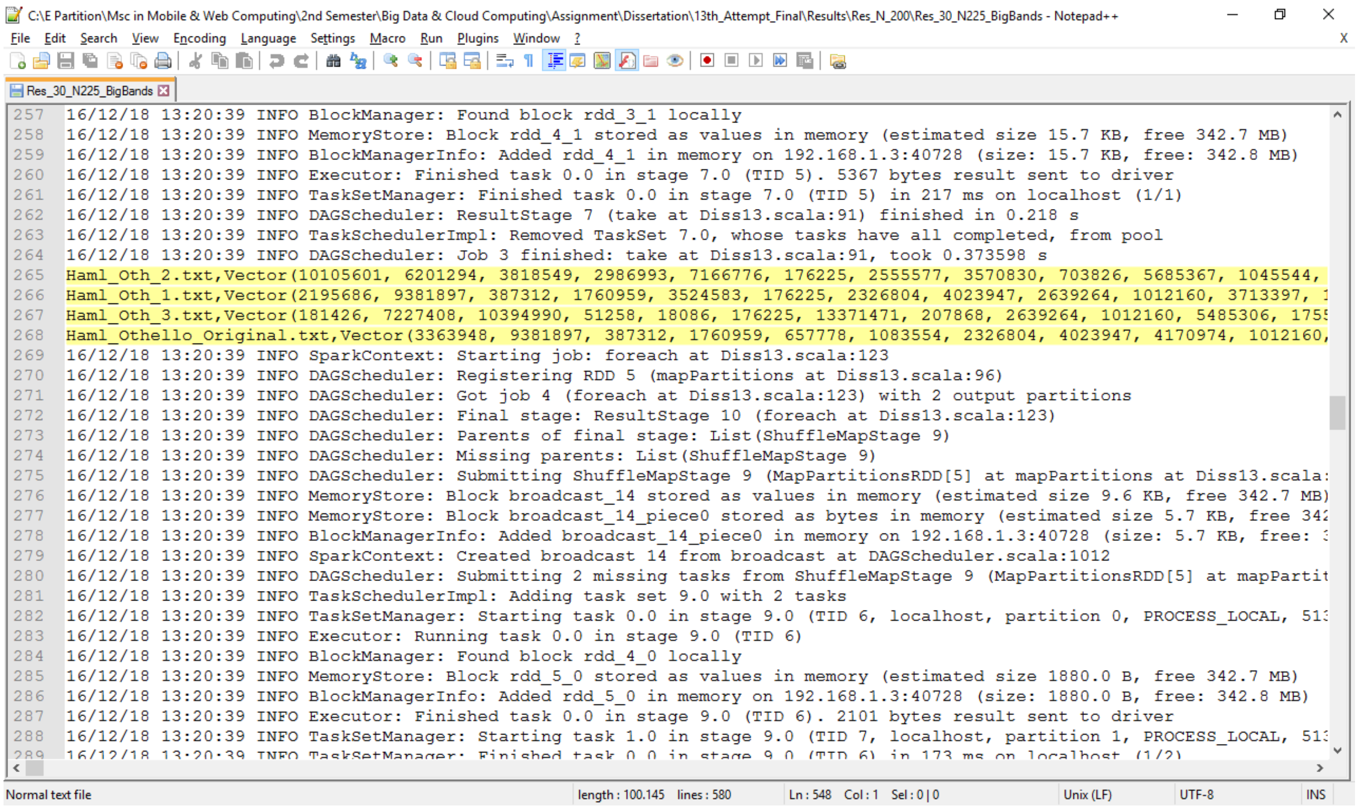
```
16/12/18 13:19:23 INFO SparkContext: Created broadcast 2 from broadcast at Diss13.scala:630
16/12/18 13:19:24 INFO MemoryStore: Block broadcast_3 stored as values in memory (estimated size 128.1 KB, free 342.8 MB
16/12/18 13:19:24 INFO MemoryStore: Block broadcast_3_piece0 stored as bytes in memory (estimated size 14.8 KB, free 342
16/12/18 13:19:24 INFO BlockManagerInfo: Added broadcast_3_piece0 in memory on 192.168.1.3:40728 (size: 14.8 KB, free: 3
16/12/18 13:19:24 INFO SparkContext: Created broadcast 3 from wholeTextFiles at Diss13.scala:640

Please Insert a Non-Prime Integer Number of Hash Functions : e.g [175 or 200]
225
16/12/18 13:20:21 INFO MemoryStore: Block broadcast_4 stored as values in memory (estimated size 24.0 B, free 342.8 MB)

Please Insert a Double Number < 1.0 for the Similarity thresshold : e.x : 0.60
16/12/18 13:20:21 INFO MemoryStore: Block broadcast_4_piece0 stored as bytes in memory (estimated size 45.0 B, free 342.
16/12/18 13:20:21 INFO BlockManagerInfo: Added broadcast_4_piece0 in memory on 192.168.1.3:40728 (size: 45.0 B, free: 34
16/12/18 13:20:21 INFO SparkContext: Created broadcast 4 from broadcast at Diss13.scala:649
0.53
16/12/18 13:20:28 INFO MemoryStore: Block broadcast_5 stored as values in memory (estimated size 24.0 B, free 342.8 MB)
16/12/18 13:20:28 INFO MemoryStore: Block broadcast_5_piece0 stored as bytes in memory (estimated size 51.0 B, free 342.
16/12/18 13:20:28 INFO BlockManagerInfo: Added broadcast_5_piece0 in memory on 192.168.1.3:40728 (size: 51.0 B, free: 34
16/12/18 13:20:28 INFO SparkContext: Created broadcast 5 from broadcast at Diss13.scala:653

Do you prefer Accuracy : B1[_ _ _], ... , B140[_ _ _]  [1]
or
Speed : B1[_ _ _ _ _ _], ... , B70[_ _ _ _ _ _]  [2]?
2
16/12/18 13:20:35 INFO MemoryStore: Block broadcast_6 stored as values in memory (estimated size 24.0 B, free 342.8 MB)
16/12/18 13:20:35 INFO MemoryStore: Block broadcast_6_piece0 stored as bytes in memory (estimated size 44.0 B, free 342.
16/12/18 13:20:35 INFO BlockManagerInfo: Added broadcast_6_piece0 in memory on 192.168.1.3:40728 (size: 44.0 B, free: 34
16/12/18 13:20:35 INFO SparkContext: Created broadcast 6 from broadcast at Diss13.scala:661
16/12/18 13:20:35 INFO SparkContext: Starting job: take at Diss13.scala:665
16/12/18 13:20:35 INFO FileInputFormat: Total input paths to process : 4
16/12/18 13:20:35 INFO FileInputFormat: Total input paths to process : 4
16/12/18 13:20:35 INFO CombineFileInputFormat: DEBUG: Terminated node allocation with : CompletedNodes: 1, size left: 13
16/12/18 13:20:36 INFO DAGScheduler: Registering RDD 1 (wholeTextFiles at Diss13.scala:640)
```


C:\E Partition\Msc in Mobile & Web Computing\2nd Semester\Big Data & Cloud Computing\Assignment\Dissertation\13th_Attempt_Final\Results\Res_N_200\Res_30_N225_BigBands - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Res_30_N225_BigBands
Normal text file
length : 100.145 lines : 580
Ln : 548 Col : 1 Sel : 0 | 0
Unix (LF)
UTF-8
INS

```
16/12/18 13:20:38 INFO ShuffleBlockFetcherIterator: Getting 2 non-empty blocks out of 2 blocks
16/12/18 13:20:38 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 1 ms
16/12/18 13:20:38 INFO MemoryStore: Block rdd_3_1 stored as bytes in memory (estimated size 12.3 KB, free 342.7 MB)
16/12/18 13:20:38 INFO BlockManagerInfo: Added rdd_3_1 in memory on 192.168.1.3:40728 (size: 12.3 KB, free: 342.9 MB)
16/12/18 13:20:38 INFO Executor: Finished task 0.0 in stage 3.0 (TID 3). 14687 bytes result sent to driver
16/12/18 13:20:38 INFO DAGScheduler: ResultStage 3 (take at Diss13.scala:665) finished in 0.218 s
16/12/18 13:20:38 INFO DAGScheduler: Job 1 finished: take at Diss13.scala:665, took 0.357737 s
Haml_Oth_2.txt,Vector(55701401, 301582874, 1822317838, 703942433, 986936866, 277514948, 562958157, 457499951, 73835523,
Haml_Oth_1.txt,Vector(55701401, 301582874, 1822317838, 703942433, 986936866, 277514948, 562958157, 457499951, 73835523,
Haml_Oth_3.txt,Vector(55701401, 301582874, 1822317838, 703942433, 986936866, 277514948, 562958157, 457499951, 73835523,
Haml_Othello_Original.txt,Vector(624272011, 2020738562, 1079283195, 2061947100, 40935353, 277684182, 238105563, 2598284
16/12/18 13:20:38 INFO TaskSetManager: Finished task 0.0 in stage 3.0 (TID 3) in 232 ms on localhost (1/1)
16/12/18 13:20:38 INFO TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have all completed, from pool
16/12/18 13:20:38 INFO MemoryStore: Block broadcast_10 stored as values in memory (estimated size 8.4 KB, free 342.7 MB)
16/12/18 13:20:38 INFO MemoryStore: Block broadcast_10_piece0 stored as bytes in memory (estimated size 2.2 KB, free 342
16/12/18 13:20:38 INFO BlockManagerInfo: Added broadcast_10_piece0 in memory on 192.168.1.3:40728 (size: 2.2 KB, free: 3
Estimated S_Curve : 1.016/12/18 13:20:38 INFO SparkContext: Created broadcast 10 from broadcast at Diss13.scala:671

Previous Rows Per Band : 225
Current Rows Per Band : 225
Number of Bands : 1


Estimated S_Curve : 0.9854585985762907
Previous Rows Per Band : 225
Current Rows Per Band : 75
Number of Bands : 3


Estimated S_Curve : 0.9648667337285327
Previous Rows Per Band : 75
Current Rows Per Band : 45
Number of Bands : 5
```

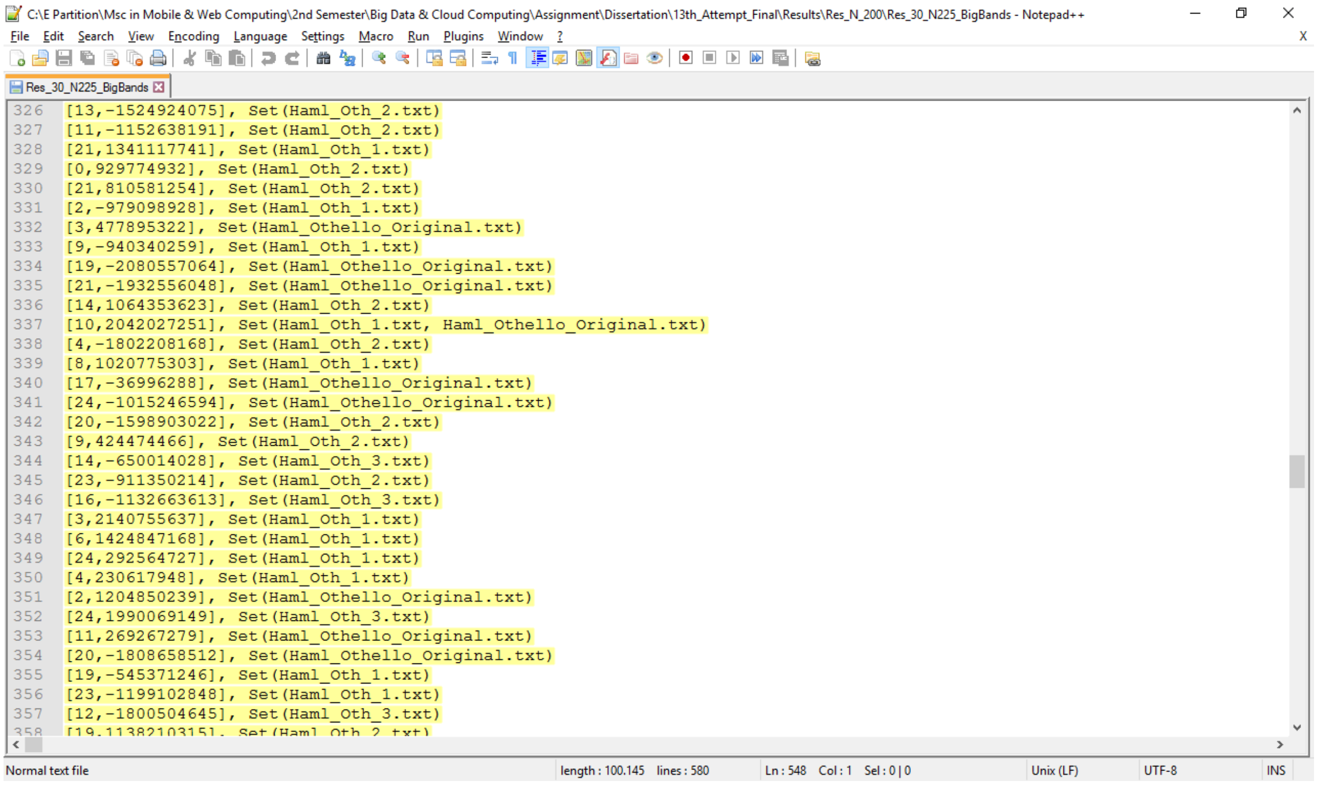

C:\E Partition\Msc in Mobile & Web Computing\2nd Semester\Big Data & Cloud Computing\Assignment\Dissertation\13th_Attempt_Final\Results\Res_N_200\Res_30_N225_BigBands - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Res_30_N225_BigBands
Normal text file
length : 100.145 lines : 580
Ln : 548 Col : 1 Sel : 0 | 0
Unix (LF)
UTF-8
INS

```

Estimated S_Curve : 0.46704367745113423
Previous Rows Per Band : 9
Current Rows Per Band : 5
Number of Bands : 45


16/12/18 13:20:38 INFO MemoryStore: Block broadcast_11 stored as values in memory (estimated size 24.0 B, free 342.7 MB)
16/12/18 13:20:38 INFO MemoryStore: Block broadcast_11_piece0 stored as bytes in memory (estimated size 44.0 B, free 342
16/12/18 13:20:38 INFO BlockManagerInfo: Added broadcast_11_piece0 in memory on 192.168.1.3:40728 (size: 44.0 B, free: 3
16/12/18 13:20:38 INFO SparkContext: Created broadcast 11 from broadcast at Diss13.scala:675

***********************************************************************************************************************
Path : /media/pankalos/96AC55CEAC55AA0F/E Partition/Msc in Mobile & Web Computing/2nd Semester/Big Data & Cloud Computir
Number of Partitions : 2
Shingle Size : 9

Number of Hashes : 225
Number of Bands : 25
Number of Rows per Band : 9
For Similarity thresshold : [0.53], user preffered : Speed
***********************************************************************************************************************

16/12/18 13:20:38 INFO SparkContext: Starting job: take at Diss13.scala:91
16/12/18 13:20:38 INFO DAGScheduler: Got job 2 (take at Diss13.scala:91) with 1 output partitions
16/12/18 13:20:38 INFO DAGScheduler: Final stage: ResultStage 5 (take at Diss13.scala:91)
16/12/18 13:20:38 INFO DAGScheduler: Parents of final stage: List(ShuffleMapStage 4)
16/12/18 13:20:38 INFO DAGScheduler: Missing parents: List()
16/12/18 13:20:38 INFO DAGScheduler: Submitting ResultStage 5 (MapPartitionsRDD[4] at mapPartitions at Diss13.scala:87),
16/12/18 13:20:38 INFO MemoryStore: Block broadcast_12 stored as values in memory (estimated size 8.2 KB, free 342.7 MB)
16/12/18 13:20:38 INFO MemoryStore: Block broadcast_12_piece0 stored as bytes in memory (estimated size 5.0 KB, free 342
16/12/18 13:20:38 INFO BlockManagerInfo: Added broadcast_12_piece0 in memory on 192.168.1.3:40728 (size: 5.0 KB, free: 3
16/12/18 13:20:38 INFO SparkContext: Created broadcast 12 from broadcast at DAGScheduler.scala:1012
```

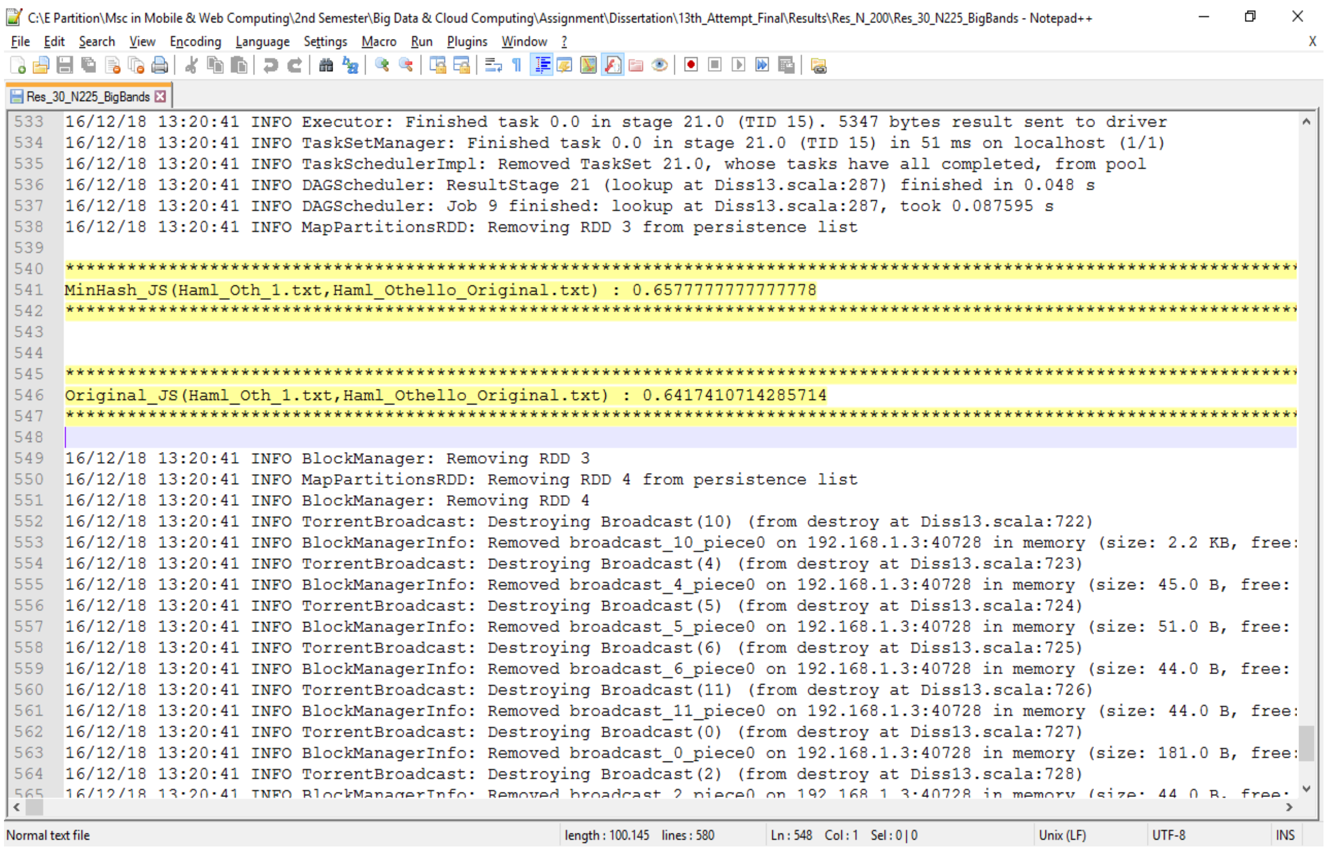

C:\E Partition\Msc in Mobile & Web Computing\2nd Semester\Big Data & Cloud Computing\Assignment\Dissertation\13th_Attempt_Final\Results\Res_N_200\Res_30_N225_BigBands - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Res_30_N225_BigBands
Normal text file
length : 100.145 lines : 580
Ln : 548 Col : 1 Sel : 0 | 0
Unix (LF)
UTF-8
INS

```
16/12/18 13:20:39 INFO BlockManager: Found block rdd_3_1 locally
16/12/18 13:20:39 INFO MemoryStore: Block rdd_4_1 stored as values in memory (estimated size 15.7 KB, free 342.7 MB)
16/12/18 13:20:39 INFO BlockManagerInfo: Added rdd_4_1 in memory on 192.168.1.3:40728 (size: 15.7 KB, free: 342.8 MB)
16/12/18 13:20:39 INFO Executor: Finished task 0.0 in stage 7.0 (TID 5). 5367 bytes result sent to driver
16/12/18 13:20:39 INFO TaskSetManager: Finished task 0.0 in stage 7.0 (TID 5) in 217 ms on localhost (1/1)
16/12/18 13:20:39 INFO DAGScheduler: ResultStage 7 (take at Diss13.scala:91) finished in 0.218 s
16/12/18 13:20:39 INFO TaskSchedulerImpl: Removed TaskSet 7.0, whose tasks have all completed, from pool
16/12/18 13:20:39 INFO DAGScheduler: Job 3 finished: take at Diss13.scala:91, took 0.373598 s
Haml_Oth_2.txt,Vector(10105601, 6201294, 3818549, 2986993, 7166776, 176225, 2555577, 3570830, 703826, 5685367, 1045544,
Haml_Oth_1.txt,Vector(2195686, 9381897, 387312, 1760959, 3524583, 176225, 2326804, 4023947, 2639264, 1012160, 3713397, 1
Haml_Oth_3.txt,Vector(181426, 7227408, 10394990, 51258, 18086, 176225, 13371471, 207868, 2639264, 1012160, 5485306, 175
Haml_Othello_Original.txt,Vector(3363948, 9381897, 387312, 1760959, 657778, 1083554, 2326804, 4023947, 4170974, 1012160,
16/12/18 13:20:39 INFO SparkContext: Starting job: foreach at Diss13.scala:123
16/12/18 13:20:39 INFO DAGScheduler: Registering RDD 5 (mapPartitions at Diss13.scala:96)
16/12/18 13:20:39 INFO DAGScheduler: Got job 4 (foreach at Diss13.scala:123) with 2 output partitions
16/12/18 13:20:39 INFO DAGScheduler: Final stage: ResultStage 10 (foreach at Diss13.scala:123)
16/12/18 13:20:39 INFO DAGScheduler: Parents of final stage: List(ShuffleMapStage 9)
16/12/18 13:20:39 INFO DAGScheduler: Missing parents: List(ShuffleMapStage 9)
16/12/18 13:20:39 INFO DAGScheduler: Submitting ShuffleMapStage 9 (MapPartitionsRDD[5] at mapPartitions at Diss13.scala:
16/12/18 13:20:39 INFO MemoryStore: Block broadcast_14 stored as values in memory (estimated size 9.6 KB, free 342.7 MB)
16/12/18 13:20:39 INFO MemoryStore: Block broadcast_14_piece0 stored as bytes in memory (estimated size 5.7 KB, free 342
16/12/18 13:20:39 INFO BlockManagerInfo: Added broadcast_14_piece0 in memory on 192.168.1.3:40728 (size: 5.7 KB, free: 3
16/12/18 13:20:39 INFO SparkContext: Created broadcast 14 from broadcast at DAGScheduler.scala:1012
16/12/18 13:20:39 INFO DAGScheduler: Submitting 2 missing tasks from ShuffleMapStage 9 (MapPartitionsRDD[5] at mapPartit
16/12/18 13:20:39 INFO TaskSchedulerImpl: Adding task set 9.0 with 2 tasks
16/12/18 13:20:39 INFO TaskSetManager: Starting task 0.0 in stage 9.0 (TID 6, localhost, partition 0, PROCESS_LOCAL, 513
16/12/18 13:20:39 INFO Executor: Running task 0.0 in stage 9.0 (TID 6)
16/12/18 13:20:39 INFO BlockManager: Found block rdd_4_0 locally
16/12/18 13:20:39 INFO MemoryStore: Block rdd_5_0 stored as values in memory (estimated size 1880.0 B, free 342.7 MB)
16/12/18 13:20:39 INFO BlockManagerInfo: Added rdd_5_0 in memory on 192.168.1.3:40728 (size: 1880.0 B, free: 342.8 MB)
16/12/18 13:20:39 INFO Executor: Finished task 0.0 in stage 9.0 (TID 6). 2101 bytes result sent to driver
16/12/18 13:20:39 INFO TaskSetManager: Starting task 1.0 in stage 9.0 (TID 7, localhost, partition 1, PROCESS_LOCAL, 513
16/12/18 13:20:39 INFO TaskSetManager: Finished task 0.0 in stage 9.0 (TID 6) in 173 ms on localhost (1/2)
```

You can notice lesser collisions to buckets from the following picture as the number of rows per bucket is bigger from the previous example.


C:\E Partition\Msc in Mobile & Web Computing\2nd Semester\Big Data & Cloud Computing\Assignment\Dissertation\13th_Attempt_Final\Results\Res_N_200\Res_30_N225_BigBands - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Res_30_N225_BigBands
Normal text file
length : 100.145 lines : 580
Ln : 548 Col : 1 Sel : 0 | 0
Unix (LF)
UTF-8
INS

```
[13,-1524924075], Set(Haml_Oth_2.txt)
[11,-1152638191], Set(Haml_Oth_2.txt)
[21,1341117741], Set(Haml_Oth_1.txt)
[0,929774932], Set(Haml_Oth_2.txt)
[21,810581254], Set(Haml_Oth_2.txt)
[2,-979098928], Set(Haml_Oth_1.txt)
[3,477895322], Set(Haml_Othello_Original.txt)
[9,-940340259], Set(Haml_Oth_1.txt)
[19,-2080557064], Set(Haml_Othello_Original.txt)
[21,-1932556048], Set(Haml_Othello_Original.txt)
[14,1064353623], Set(Haml_Oth_2.txt)
[10,2042027251], Set(Haml_Oth_1.txt, Haml_Othello_Original.txt)
[4,-1802208168], Set(Haml_Oth_2.txt)
[8,1020775303], Set(Haml_Oth_1.txt)
[17,-36996288], Set(Haml_Othello_Original.txt)
[24,-1015246594], Set(Haml_Othello_Original.txt)
[20,-1598903022], Set(Haml_Oth_2.txt)
[9,424474466], Set(Haml_Oth_2.txt)
[14,-650014028], Set(Haml_Oth_3.txt)
[23,-911350214], Set(Haml_Oth_2.txt)
[16,-1132663613], Set(Haml_Oth_3.txt)
[3,2140755637], Set(Haml_Oth_1.txt)
[6,1424847168], Set(Haml_Oth_1.txt)
[24,292564727], Set(Haml_Oth_1.txt)
[4,230617948], Set(Haml_Oth_1.txt)
[2,1204850239], Set(Haml_Othello_Original.txt)
[24,1990069149], Set(Haml_Oth_3.txt)
[11,269267279], Set(Haml_Othello_Original.txt)
[20,-1808658512], Set(Haml_Othello_Original.txt)
[19,-545371246], Set(Haml_Oth_1.txt)
[23,-1199102848], Set(Haml_Oth_1.txt)
[12,-1800504645], Set(Haml_Oth_3.txt)
[19,1138210315], Set(Haml_Oth_2.txt)
```


C:\E Partition\Msc in Mobile & Web Computing\2nd Semester\Big Data & Cloud Computing\Assignment\Dissertation\13th_Attempt_Final\Results\Res_N_200\Res_30_N225_BigBands - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Res_30_N225_BigBands
Normal text file
length : 100.145 lines : 580
Ln : 548 Col : 1 Sel : 0 | 0
Unix (LF)
UTF-8
INS

```
16/12/18 13:20:41 INFO Executor: Finished task 0.0 in stage 21.0 (TID 15). 5347 bytes result sent to driver
16/12/18 13:20:41 INFO TaskSetManager: Finished task 0.0 in stage 21.0 (TID 15) in 51 ms on localhost (1/1)
16/12/18 13:20:41 INFO TaskSchedulerImpl: Removed TaskSet 21.0, whose tasks have all completed, from pool
16/12/18 13:20:41 INFO DAGScheduler: ResultStage 21 (lookup at Diss13.scala:287) finished in 0.048 s
16/12/18 13:20:41 INFO DAGScheduler: Job 9 finished: lookup at Diss13.scala:287, took 0.087595 s
16/12/18 13:20:41 INFO MapPartitionsRDD: Removing RDD 3 from persistence list

*****************************************************************************************************************************
MinHash_JS(Haml_Oth_1.txt,Haml_Othello_Original.txt) : 0.6577777777777778
*****************************************************************************************************************************


*****************************************************************************************************************************
Original_JS(Haml_Oth_1.txt,Haml_Othello_Original.txt) : 0.6417410714285714
*****************************************************************************************************************************

16/12/18 13:20:41 INFO BlockManager: Removing RDD 3
16/12/18 13:20:41 INFO MapPartitionsRDD: Removing RDD 4 from persistence list
16/12/18 13:20:41 INFO BlockManager: Removing RDD 4
16/12/18 13:20:41 INFO TorrentBroadcast: Destroying Broadcast(10) (from destroy at Diss13.scala:722)
16/12/18 13:20:41 INFO BlockManagerInfo: Removed broadcast_10_piece0 on 192.168.1.3:40728 in memory (size: 2.2 KB, free:
16/12/18 13:20:41 INFO TorrentBroadcast: Destroying Broadcast(4) (from destroy at Diss13.scala:723)
16/12/18 13:20:41 INFO BlockManagerInfo: Removed broadcast_4_piece0 on 192.168.1.3:40728 in memory (size: 45.0 B, free:
16/12/18 13:20:41 INFO TorrentBroadcast: Destroying Broadcast(5) (from destroy at Diss13.scala:724)
16/12/18 13:20:41 INFO BlockManagerInfo: Removed broadcast_5_piece0 on 192.168.1.3:40728 in memory (size: 51.0 B, free:
16/12/18 13:20:41 INFO TorrentBroadcast: Destroying Broadcast(6) (from destroy at Diss13.scala:725)
16/12/18 13:20:41 INFO BlockManagerInfo: Removed broadcast_6_piece0 on 192.168.1.3:40728 in memory (size: 44.0 B, free:
16/12/18 13:20:41 INFO TorrentBroadcast: Destroying Broadcast(11) (from destroy at Diss13.scala:726)
16/12/18 13:20:41 INFO BlockManagerInfo: Removed broadcast_11_piece0 on 192.168.1.3:40728 in memory (size: 44.0 B, free:
16/12/18 13:20:41 INFO TorrentBroadcast: Destroying Broadcast(0) (from destroy at Diss13.scala:727)
16/12/18 13:20:41 INFO BlockManagerInfo: Removed broadcast_0_piece0 on 192.168.1.3:40728 in memory (size: 181.0 B, free:
16/12/18 13:20:41 INFO TorrentBroadcast: Destroying Broadcast(2) (from destroy at Diss13.scala:728)
16/12/18 13:20:41 INFO BlockManagerInfo: Removed broadcast_2_piece0 on 192.168.1.3:40728 in memory (size: 44.0 B, free:
```

7.3 Source code

For quick review of the source code please refer to files:

- Quick Review of the Source Files/Diss13.scala
- Quick Review of the Source Files/built.sbt

To run the program:

- Open Diss13 project with IntelijIDEA (Note that, you must have installed IntelijIDEA with Spark and Scala), I run it in Ubuntu 16.04 but in Windows should be fine too.

  Or:

- To compile and Run the program without an IDEA, (Note that you must have a system with Spark and Scala installed) do the following :

From Linux:

Get in Diss13 Directory:

1) $cd Diss13/
2) $sbt package

- [info] Packaging {..}/{..} Diss13/target/scala-2.10/diss13_2.10-1.0.jar

Then run it by:

3) $YOUR_SPARK_HOME/bin/spark-submit ./{..}/{..}Diss13/target/scala-2.10/diss13_2.10-1.0.jar

I have an assembled diss13_2.10-1.0.jar package in [Quick Review of the Source Files/] Directory, so we are ready to run it.

You can also find the examples in Directory:

- 3_Oth/

And more results in Directory:

- Results/