

**MINI - PROJECT REPORT ON**

**“Mini Project Plagiarism Detector”**

**BY**

Tanmay Joshi

&

Rohan Bhange

**Under the Guidance of :**

***Prof. Kiran Bidua***

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

**MIT SCHOOL of Engineering**

Loni Kalbhor Pune

1

**M.I.T. SCHOOL OF ENGINEERING**

DEPARTMENT OF COMPUTER ENGINEERING

LONI – KALBHOR PUNE

***CERTIFICATE***

******

This is to certify that the Mini- Project report entitled

“*MINI PROJECT PLAGIARISM DETECTOR*”

submitted by

**Tanmay Joshi MITU20BTCSD035**

**Rohan Bhange MITU19BTML0023**

is a record of Bonafede work carried out by them, under my guidance, in partial fulfilment of the requirement for the Second Year of Engineering (Computer) at M.I.T. School of Engineering, Pune under MIT Art, Design & Technology University.

Date: Place:

**Prof. Kiran Bidua Dr. RajneeshKaur Sachdeo**

**Guide, Dean Engineering,**

**Department of CSE Head, Department of CSE M.I.T. School of Engineering MIT School of Engineering Loni Kalbhor, Pune Loni-Kalbhor, Pune**

2

**ACKNOWLEDGEMENT**

We would like to express our humble gratitude towards our mentor Prof.Kiran Bidua as well as our principal Dr. Kishore Ravande who gave us this golden opportunity to work on this interesting project helped us gain in depth knowledge about natural language processing which is the future of this fast-paced world. It also helped analyse the Pros and Cons of a particular application and its market viability. We provide valuable information through our analysis which will help upcoming app developers plan meticulously as to how their application has to be designed. Both members contributed greatly to the timely completion of this project which would have been difficult without the assistance of our guide.

3

**TABLE OF CONTENTS**

| **SR**  **NO** | **CONTENTS PAGE NO** |
| --- | --- |
| **1** | **5**  **ABSTRACT** |
| **2** | **LIST OF TABLES** |
| **3** | **LIST OF FIGURES** |
| **4** | 1. **INTRODUCTION**  **6**  **7**  2. **PROBLEM DEFINITION**  **7**  3. **FEATURES OF PROJECT**  **8**  4. **PLATFORM / TECHNOLOGY**  **9**  5. **FLOWCHART**  6. **OUTPUT**  **10**  **24**  7. **CODE**  8. **GLOSSARY**  **48**  9. **REFERENCES**  10. **CONCLUSION AND FUTURE**  **49**  **ENHANCEMENTS** |
| **5** | **GLOSSARY** |
| **6** | **REFERENCE AND BIBLIOGRAPHY 48** |
| **7** | **50**  **ANNEXURE A** |

4

**ABSTRACT**

* To protect users from numerous password inference attacks, we invent a novel context aware privacy enhancing keyboard (Xboard) for Android touch-based devices. Usually Xboard would show a QWERTY keyboard when users input text like an email or a message. Nevertheless, whenever users enter a password in the input box on his or her touch-enabled device, a keyboard will be shown to them with the positions of the characters shuffled at random.

5

**INTRODUCTION**

Our project focuses on creating a secure keyboard which deals with the issues of privacy, data breach.​

A high fidelity UI keyboard, with numerous functions for customization and ease of use will be implemented.​

Beyond its appearance and customization features it will be loaded with the security features unlike no other keyboards promises in the market.​

Cryptography algorithms will be implemented and it will be equipped with an option to choose type of encryption based on the level of security they provide for secure messaging.​

6

**PROBLEM STATEMENT**

**Let's Talk about messing and texting.... now a days there is a large number of youngsters use texting apps like WhatsApp, Instagram etc..**​

**Let's say you are texting, you are sending some messages and you will want that the message is only read by the person intended to. And have "received" exactly what you have "sent".**​

​

**OfCourse it is ensured in Face-to-Face case.**​

​

**Now we have a question : What if the channel you are communicating through is under the influence of a bad person or a third person that can and will manipulate your conversation.**

**FEATURES OF THE PROJECT**

The project has the following basic features:

**1. Flexible:**​

**Works in all of your favorite messaging apps like WhatsApp, Signal,** ​

**Messenger, etc.**​

**2. Cryptography:**​

**Secure all your messages using AES(Advanced Encryption Standard) algorithm.**​

**3.Cryptography:**​

**Decrypt all your messages using same algorithm AES(Advanced Encryption Standard).**​

**4.Usability:**​

**Simple to use, user friendly UI design.**​

**5.One tap Encryption:** ​

**Encrypt or Decrypt all your messages that you "Sent" or "Received" in a single tap of a button.**​

**6. Send Secure Emails:**​

**Send your Important files and messages under a secured channel.**

7

**PLATFORM / TECHNOLOGY USED IN THE PROJECT**

**1. JAVA**

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let application developers write once, run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture.

**Libraries used:**

1.commons math3 3.6.1

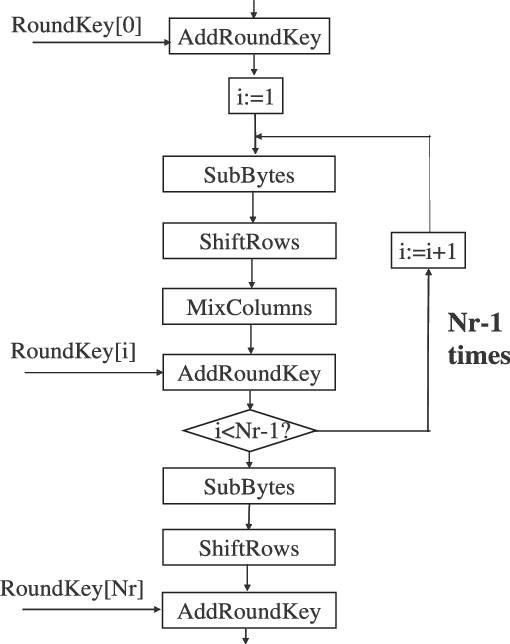
2.mysql connector java 8.0.25

3.opennlp 1.9.3

4.pdfbox 2.0.24

8

**FLOWCHART:**

9

**OUTPUT**

The Login Screen.

|  |
| --- |

After Sign up:

|  |
| --- |

Select Keyboard :

|  |
| --- |

Choose Xboard as default keyboard :

|  |
| --- |

Encryption Message :

|  |
| --- |

12

Decryption Message :

|  |
| --- |

13

***BACKEND CODE PROCESSES:***

Step by step description.

**Text pre-processing.**

text pre-processing involves the following steps:

**Punctuation removal**:

In this step all punctuation marks are removed from the text. The remove\_punc function handles this process.

static String remove\_punc(String content) {

content = content.replaceAll("\\p{Punct}", "");

return content;

}

here, \\p{Punct} is the regex for punctuation marks which are replaced by the empty string.

**Stopword Removal:**

stop words are words which are filtered out before or after processing of natural language data (text). These are actually the most common words in any language (like articles, prepositions, pronouns, conjunctions, etc) and *does not add much information to the text*. Examples of a few stop words in English are “the”, “a”, “an”, “so”, “what”.By removing these words, we remove the low-level information from our text in order to give more focus to the important information.

static String[] remove\_stopwords(String content) throws IOException {

ArrayList<String> stopwords;

stopwords = (ArrayList<String>) Files.*readAllLines*(Paths.*get*("src/sample/stopwords.txt")); ArrayList<String> allWords1 = Stream.*of*(content.toLowerCase().split(" "))

.collect(Collectors.*toCollection*(ArrayList<String>::new));

allWords1.removeAll(stopwords);

return allWords1.toArray(new String[0]);

}

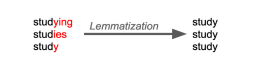
Here the String the holding the content read from a file is split along the whitespace character and then converted to an ArrayList object. The stopwords are read from a file in the directory and is stored in another arrayList object. The

14

removeAll method is finally called and all stopwords are removed, the final arrayList is returned as an Array of Strings.

**Lemmatization:**

Lemmatization is an algorithmic process of determining and reducing the word to its lemma based on its meaning and context the word is used in. All inflected forms of a word are reduced to their lemma so that they can be analysed as one word. In order to perform lemmatization the context and meaning of the words need to be understood which is done via Part Of Speech tagging. The tags for words are nouns, verbs, adjectives etc. Words along with their tags are then used for lemmatization.



static String[] POS\_tagging(String[] tokens) throws IOException {

InputStream inputStreamPOSTagger = new FileInputStream("src/sample/en-pos-maxent.bin"); POSModel posModel = new POSModel(inputStreamPOSTagger);

POSTaggerME posTagger = new POSTaggerME(posModel);

return posTagger.tag(tokens);

}

this function accepts the Array of words contained in the file and determines their POS tag using a file that contains English words with their corresponding tags. The tags for the corresponding words are then returned by the function.

static String[] lemmatization(String[] tags, String[] tokens) throws IOException {

String[] lemmas;

InputStream DictLemmatizer = new FileInputStream("src/sample/lemmatizer-dict.txt"); DictionaryLemmatizer lemmatizer = new DictionaryLemmatizer(DictLemmatizer);

lemmas = lemmatizer.lemmatize(tokens, tags);

for (int i = 0; i < lemmas.length; i++) {

if (lemmas[i].equals("O")) {

lemmas[i] = tokens[i];

}

}

return lemmas;

}

This function accepts the words and corresponding tags and the lemmatize method of DictionaryLemmatizer object to generate the required lemmas using the lemmatizer-dict text file that contains a list of words, POStags and the corresponding lemmas. The method however returns “O” for words (proper

15

nouns) whose lemmas don’t exist. Therefore the for loop replaces the “O” with the corresponding original proper nouns.

**Model Building**

Building a mathematical model from the data that can mathematically represent the data is the in these steps:

**Ngrams Generation:**

Ngrams of a list of words consist of all possible combinations of n consecutive words from that list. For e.g. 2-grams of the sentence *“This project detects plagiarism”* are “*this project”, “project detects”, “detects plagiarism”.* Experiments show the most suitable n-grams length lies between 2 and 7 **[1]**. Smaller values of n are unable to detect over lapping text, while larger values of n cannot detect fine grained modifications.

static String[] n\_gram(String[] tokens) {

return NGramGenerator.*generate*(Arrays.*asList*(tokens), 2, " ").toArray(new String[0]); }

this function takes the array of words that have been lemmatized and generates bi-grams and returns the bi-grams as an array.

**Vocabulary Building:**

Vocabulary is a tool that enables us to build mathematical models to represent our documents numerically. The vocabulary is a set of all unique terms (bi-grams in this case) present in all the documents. The vocabulary is an unordered set and has no repeating elements. After the n-grams for each document are generated they are added to the vocabulary.

static void create\_bow(String[] ngrams) {

*bag\_of\_words*.addAll(Arrays.*asList*(ngrams));

}

This function adds the bi-grams from each document to the bag\_of\_words Which is a static variable representing our vocabulary.

16

**Phrase Extraction:**

Since we are dealing with a large number of documents, a *feature selection* method must be employed to reduce the number of phrases to reduce processing time. The most simple and from our point of view the best method for plagiarism detection is a Document Frequency (DF) feature selection. According to the document frequency we can simply determine if the given phrase is important or not. The phrases existing just in one document are removed right away since they cannot be plagiarized in any other document.

static void document\_frequency() {

double df;

*doc\_freq* = new double[*bag\_of\_words\_arr*.length];

for (int i = 0; i < *bag\_of\_words\_arr*.length; i++) {

*doc\_freq*[i] = 0.0;

df = 0.0;

for (String[] doc : *all\_ngrams*) {

for (String s : doc) {

if (s.equals(*bag\_of\_words\_arr*[i])) {

*doc\_freq*[i] += 1;

break;

}

}

}

}

This function generates the document frequency for all terms in the vocabulary. The document frequencies are stored in a global variable called doc\_freq.

static void phrase\_extraction() {

StandardDeviation std = new StandardDeviation();

Mean m = new Mean();

System.*out*.println("printing doc frq");

for(double j:*doc\_freq*){

System.*out*.println(j);

}

double std\_dev = std.evaluate(*doc\_freq*);

double mean = m.evaluate(*doc\_freq*, 0, *doc\_freq*.length - 1);

ArrayList<String> tmp\_arr= new ArrayList<>(Arrays.*asList*(*bag\_of\_words\_arr*));

for(int i= tmp\_arr.size()-1;i>=0;i--)

{

if(*doc\_freq*[i]==1){*//|| doc\_freq[i]>std\_dev+mean*

tmp\_arr.remove(i);

}

17

}

*// to calculate phrases removed for normalization at end.*

double removed;

for(int i=0;i<*all\_ngrams*.size();i++){

removed=0.0;

for(String s: *all\_ngrams*.get(i)){

if(!tmp\_arr.contains(s)){

removed++;

}

}

*phr\_red*[i]=*ph\_org*[i]-removed;

}

*bag\_of\_words\_arr*=tmp\_arr.toArray(new String[0]);

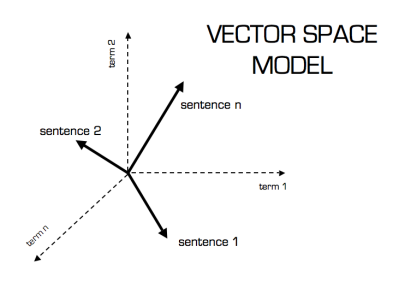
}

This function uses the document frequency to remove all words from the vocabulary that occur in only one document. The number of terms removed from each document because of feature selection is computed and stored as well for later use in normalizing the results.

18

**Vector Space Model:**

The vector space model is an algebraic model for representing text documents as vectors of terms. Each dimension of the vector space corresponds to a term. In this case the terms are the elements of our vocabulary.

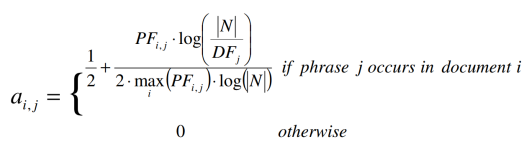


Our documents will be represented as n dimensional vectors where n is the size of the vocabulary.

**Term Frequency-Inverse Document Frequency:**

The values along each dimension for a document vector are determined using TF-IDF weighting. TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents. The formula used in this project is a light variation of the traditional one.

19

*PF i,j* represents the occurrence frequency of phrase *j* in document *i*, *DFj* represents the number of documents where phrase *j* occurs.

static double[] phrase\_frequency(String[] doc,int k) { *// divide by new length jeesus christ.* double count = 0.0;

double[] pf = new double[*bag\_of\_words\_arr*.length];

for (int j = 0; j < *bag\_of\_words\_arr*.length; j++) {

count = 0.0;

for (int i = 0; i < doc.length; i++) {

if (*bag\_of\_words\_arr*[j].equals(doc[i])) {

count++;

}

}

pf[j] = count;

}

Pharse frequency generates PF mentioned above.

static double[] tfidf\_vectorization(double[] doc\_pf) {

double[] tfidf = new double[*bag\_of\_words\_arr*.length];

double max=Arrays.*stream*(doc\_pf).max().getAsDouble();

for (int i = 0; i < *bag\_of\_words\_arr*.length; i++) {

if (doc\_pf[i] == 0.0){

tfidf[i] = 0.0;

}

else{

tfidf[i] = 0.5 + ((doc\_pf[i] \* Math.*log*(*N* / *doc\_freq*[i])) / (2 \* Math.*log*(*N*)\*max)); }

}

return tfidf;

}

20

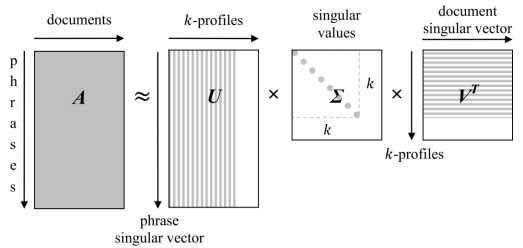
The tfidf\_vecotization function then calculates and returns the document vector using the formula mentioned above and stores it in a two dimensional matrix for further use.

**Similarity Calculation:**

The final part involves processes to calculate the similarity between document vectors.

**Latent semantic analysis:**

Latent semantic analysis (LSA) is a technique in natural language processing, in particular distributional semantics, of analysing relationships between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms. matrix containing word counts per document (rows represent unique words and columns represent each document) is constructed from a large piece of text and a mathematical technique called singular value decomposition (SVD) is used to reduce the number of rows while preserving the similarity structure among columns.

**Document Similarity Normalization:**

21

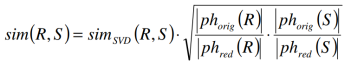
Before we use matrix VT, single elements of all document profiles must be rescaled with the corresponding singular values.

B = Σ ×VT

Finally, we compute the mutual pairwise correlation according to Equation where the columns of matrix B are length-normalized. The resulting matrix is a symmetric matrix where each pair of documents is evaluated by a score representing the percentage similarity.



simSVD obtains much higher score for such documents where vast majority of phrases are removed as meaningless. Therefore, we need to normalize the results with the following formula.



where the evaluation of documents R and S is weighted by the ratio between the number of original phrases |phorig| and the number of phrasesafter reduction |phred|.

static RealMatrix SVD(RealMatrix m) {

SingularValueDecomposition svd = new SingularValueDecomposition(m.transpose());

return svd.getS().multiply(svd.getVT());

}

static RealMatrix length\_normalize(RealMatrix b) {

int col\_d = b.getColumnDimension();

int row\_d = b.getRowDimension();

for (int i = 0; i < col\_d; ++i) {

double sum = 0.0;

double length;

for (int j = 0; j < row\_d; j++) {

*//System.out.println(B.getEntry(j,i));*

sum += b.getEntry(j, i) \* b.getEntry(j, i);

}

length = Math.*sqrt*(sum);

System.*out*.print("length is ");

System.*out*.println(length);

for (int j = 0; j < row\_d; j++) {

b.multiplyEntry(j, i, 1.0 / length);

}

}

return b;

22

}

svd gives the singular value decomposition of the document model matrix and length\_normalize function length normalizes the columns of matrix b as per the formula.

for(int j=0;j< tmp.length;j++)

{

if(*phr\_red*[i]!=0 && *phr\_red*[j]!=0) {

tmp[i][j]=tmp[i][j] \* Math.*sqrt*((*ph\_org*[i] \* *ph\_org*[j]) / (*phr\_red*[i] \* *phr\_red*[j]));

this piece of code then normalizes all the values in the similarity to get the final similarity score between two documents.

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

**References:**

[1] Ceska Z. (2008) Plagiarism Detection Based on Singular Value Decomposition. In: Nordström B., Ranta A. (eds) Advances in Natural Language Processing. GoTAL 2008. Lecture Notes in Computer Science, vol 5221. Springer, Berlin, Heidelberg.

48

**Conclusion:**

Mini Project Plagiarism Detector is a robust plagiarism detection software. The key concept is to detect the plagiarism between the inserted documents to prevent malpractices.

The user with minimum knowledge about computers can be able to operate the software easily.

We learnt a lot of new things about the NLP(Natural Language Processing), Java and Software Development, and were able to successfully implement the project in limited time using team and time management.

Future Scope:

More accurate backend algorithm can be implemented.

GUI can be enhanced.

Client Server Architecture can be implemented, and the project can be modified to the entire Mini Project Management System.

49

Mini Project Sem 4 Tracker

Mini Project Tracker (Group.: 08) Guide: Prof. Suresh Kapare Task in Progress Task Completed

**PROJECT START DATE 21-Mar-21 SCROLL TO WEEK # 1 Mar-26 Apr-2 Apr-9 Apr-16 Apr-23 Apr-30**

**Sprint Priority Start Finish Duration Status % Complete** 22 23 24 25 26 29 30 31 1 2 5 6 7 8 9 12 13 14 15 16 19 20 21 22 23 26 27 28 29 30 M T W T F M T W T F M T W T F M T W T F M T W T F M T W T F

**Project Summary 21-Mar-21 26-Jun-21 70 day(s) 100%**

**Rohan Bhange (Front End) 21-Mar-21 26-Jun-21 70 day(s)**

Research High 21-Mar-21 10-Apr-21 15 day(s) Green 100%

Learning (Language and Platform) Medium 11-Apr-21 25-Apr-21 10 day(s) Green 100%

Coding High 26-Apr-21 16-May-21 15 day(s) Green 100%

Implementation High 17-May-21 31-May-21 11 day(s) Green 100%

Front End Testing High 1-Jun-21 9-Jun-21 7 day(s) Green 100%

Integration High 10-Jun-21 20-Jun-21 7 day(s) Green 100%

Final Testing High 21-Jun-21 26-Jun-21 5 day(s) Green 100%

**Burhanuddin Unwalla (Back End) 21-Mar-21 26-Jun-21 70 day(s)**

Research High 21-Mar-21 10-Apr-21 15 day(s) Green 100%

Learning (Language and Platform) Medium 11-Apr-21 25-Apr-21 10 day(s) Green 100%

Coding High 26-Apr-21 16-May-21 15 day(s) Green 100%

Implementation High 17-May-21 31-May-21 11 day(s) Green 100%

Back End Testing High 1-Jun-21 9-Jun-21 7 day(s) Green 100%

Integration High 10-Jun-21 20-Jun-21 7 day(s) Green 100%

Final Testing High 21-Jun-21 26-Jun-21 5 day(s) Green 100%

50