Université Concordia University

# SOEN 691-UU Big Data Analytics

## Code Clone Detection


Team :

| | |
|---|---|
| Pulkit Wadhwa | 40082832 |
| Ankur Aggarwal | 40105298 |
| Jasmeet | 40088712 |

# TABLE OF CONTENTS

# 01 INTRODUCTION

- Code Clones: Similar or identical fragments of code
- Do Code Clones really matter?
  - Defect prone
  - Problem of redundancy and increase in size of program
- Motivation:
  - Detecting clones can help in decreasing maintenance cost.
  - Auto Comment Generation of programs.

# 02

# DataSet Generation

- No available Dataset of code clones, thus No Definitive features that characterize these clones
- Used IJADataset which has a collection of Java programs .
- Performed lexical analysis on these Java source codes to generate tokens for each program.
- JAVALANG tool was used for lexical Analysis
- Tokens include keyword, identifier , modifier, separator.
- Used the count of each token as features to generate the dataset.

# 02

# DataSet Generation

- DataSet Contains 56,168 rows (or programs),including 10k duplicates approx and 15 different features

```java
public class AddTwoNumbers {

   public static void main(String[] args) {

      int num1 = 5, num2 = 15, sum;
      sum = num1 + num2;

      System.out.println("Sum of these numbers: "+sum);
   }
}
```

Lexical

Analysis

```
Modifier:3
Keyword:2
Identifier:14
Separator:17
BasicType:1
Operator:5
DecimalInteger:2
String:1
```

| Keyword | Identifier | Separator | Operator | Modifier | String | Null | BasicType | DecimalInteg | Boolean | DecimalFloat | Annotation | HexInteger | OctalInteger | HexFloatingPoint |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 30 | 166 | 232 | 35 | 2 | 26 | 1 | | | | | | | | |
| 47 | 201 | 265 | 46 | 4 | 14 | 2 | 8 | 5 | | | | | | |
| 49 | 180 | 238 | 28 | 7 | 15 | 1 | 3 | | 1 | | | | | |
| 12 | 76 | 90 | 13 | 3 | 2 | 1 | 2 | 1 | 1 | | | | | |
| 33 | 177 | 280 | 41 | 6 | 12 | 9 | 4 | 11 | | | | | | |
| 19 | 124 | 177 | 23 | 3 | 4 | 3 | 1 | 7 | | | | | | |
| 19 | 124 | 177 | 23 | 3 | 4 | 3 | 1 | 7 | | | | | | |
| 19 | 124 | 177 | 23 | 3 | 4 | 3 | 1 | 7 | | | | | | |
| 75 | 366 | 589 | 86 | 5 | 41 | 26 | 2 | 5 | 6 | | | | | |
| 130 | 946 | 1445 | 214 | 23 | 153 | 39 | 19 | 13 | 12 | | | | | |

# Injecting Code Clones in Dataset

The approach we followed to add code clones in DataSet are :-

- Type 1 clones(Exact CLones)
    - Created multiple copies of the codes .
    - Addition of comments in few codes.
- Type 2 clones(Renamed CLones):
    - Modifications in identifier names including Type 1 changes.

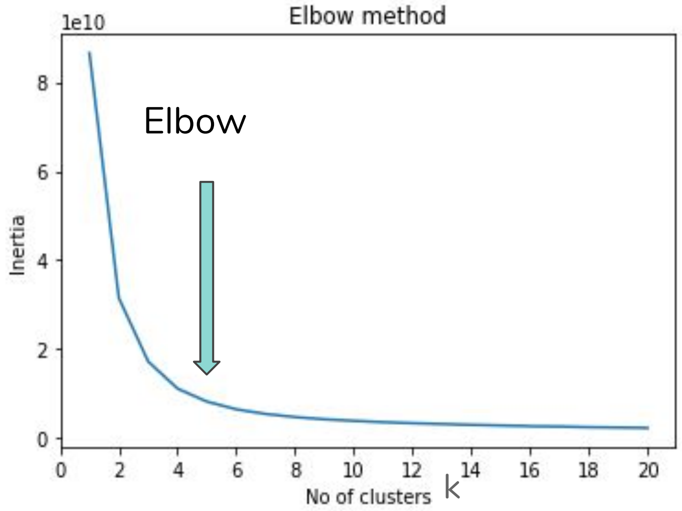
# 03

# Algorithms Used

- Challenges
    1. High dimensional feature vector representing each program
    2. Comparison of instance with all other instances in dataset for finding similarity is expensive.
- K-Means Clustering: Group the points into K clusters on the basis of distance between points.
- We used K-Means implementation of scikit-learn
- Initially we used original higher dimensional dataset for clustering

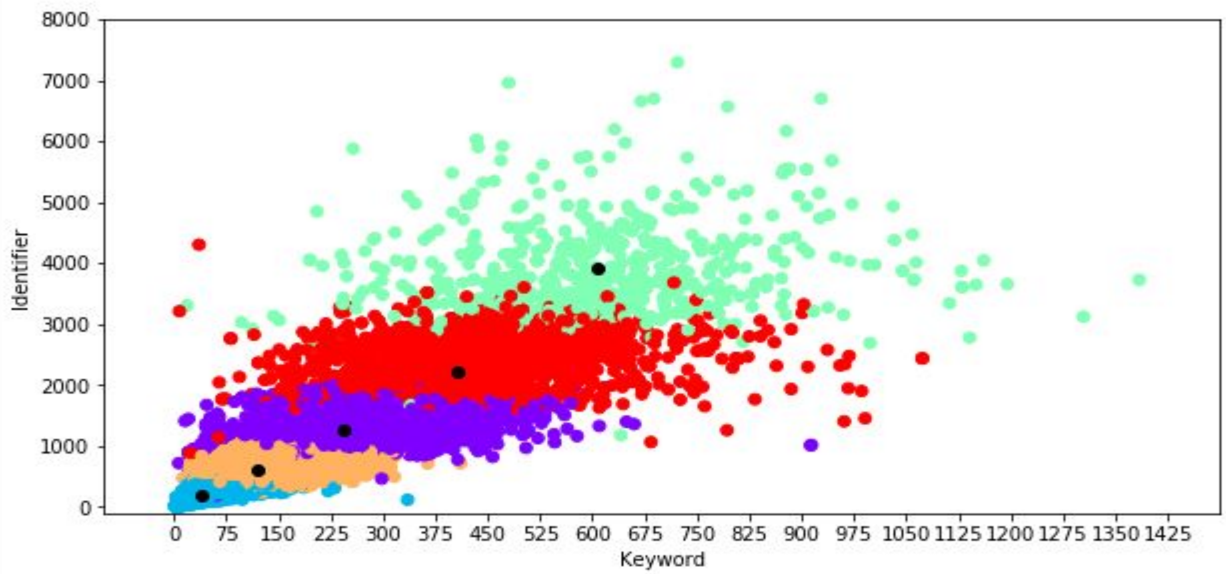
# Choosing Right Value of K

- Inertia_ :Sum of squared distances of samples to their closest cluster center.
- Aim to choose k that have a small value of inertia
- Used Elbow method for finding the right value of K.
- We can choose the elbow point k=5 as after this point change in inertia isn't signficant



# Clustering with k=5

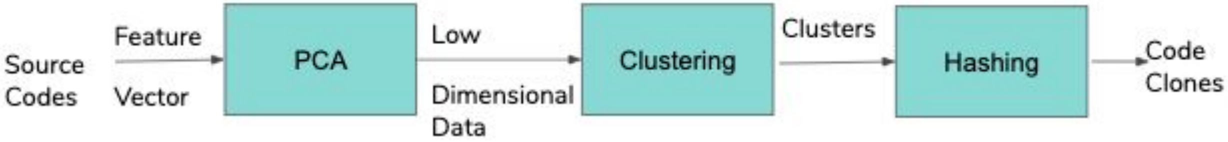
- Used k(number of clusters)=5
- Maximum iterations=10,000 and initialisation method of k-means++



# Drawbacks k=5

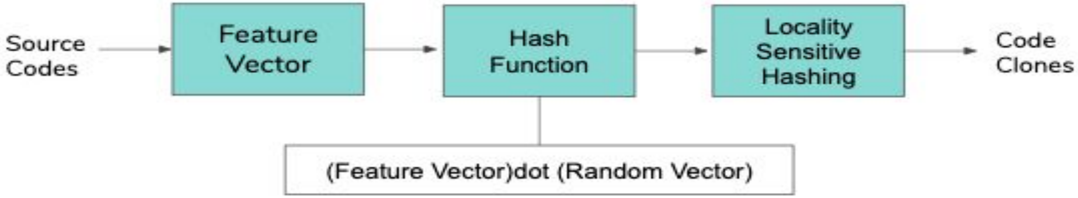
- K=5  won't be a good clone detector.
- Dataset contains 45k non duplicated programs so ideally it should have around 45k clusters.
- To deal with problem of dimensionality we also tried PCA(Principal Component Analysis ) dimensionality reduction technique.
- Results with clustering on dimensionally reduced data were same, Elbow was at k=5

# Advanced Approach 1




- To Deal with first challenge we used dimensionality reduction technique.
- Second can be addressed using a combination of clustering and nearest neighbour search.
- Clustering(k=5) helped to divide input space into smaller subspaces.
- Next step was to find nearest neighbours in these subspaces.
- Used Hashing as a candidate for nearest neighbour search.

# Locality Sensitive Hashing using Random Projection




- Locality Sensitive Hashing solves both the problems.
- The first Hash function reduces the dimensionality of the dataset.
- Further second Hash Function gives the exact similar pairs we use band=1
- As problem demanded to find the exact similar items instead of finding the candidate pairs for similarity

# 04

# Performance

- Compared the performance of both the approaches on the basis of execution time.
- Locality sensitive Hashing detected duplicates faster than using Clustering and Hashing together.

| Approach | Execution TIme |
| --- | --- |
| PCA,K-Means Clustering and Hashing | 3.71 Seconds |
| Locality Sensitive Hashing Using Random Projection | 2.16 Seconds |
| Locality Sensitive Hashing Using Gaussian Projection(Scikit learn) | 1.82 Seconds |

# 05

# Future Work

- Extending our work for Semantic clones.
- Including more programming languages.
- Parallelized implementation of Locality Sensitive Hashing.

# REFERENCES


- https://towardsdatascience.com/locality-sensitive-hashing-for-music-search-f2f1940ace23
- https://heartbeat.fritz.ai/k-means-clustering-using-sklearn-and-python-4a054d67b187
- https://medium.com/analytics-vidhya/how-to-determine-the-optimal-k-for-k-means-708505d204eb
- http://courses.cs.vt.edu/cs5704/spring16/handouts/5704-10-CodeClones.pdf

# Thank you!