Alma Mater Studiorum - Università di Bologna

Big Data Project - Presentation
Textile Defect Detection

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Artificial Intelligence

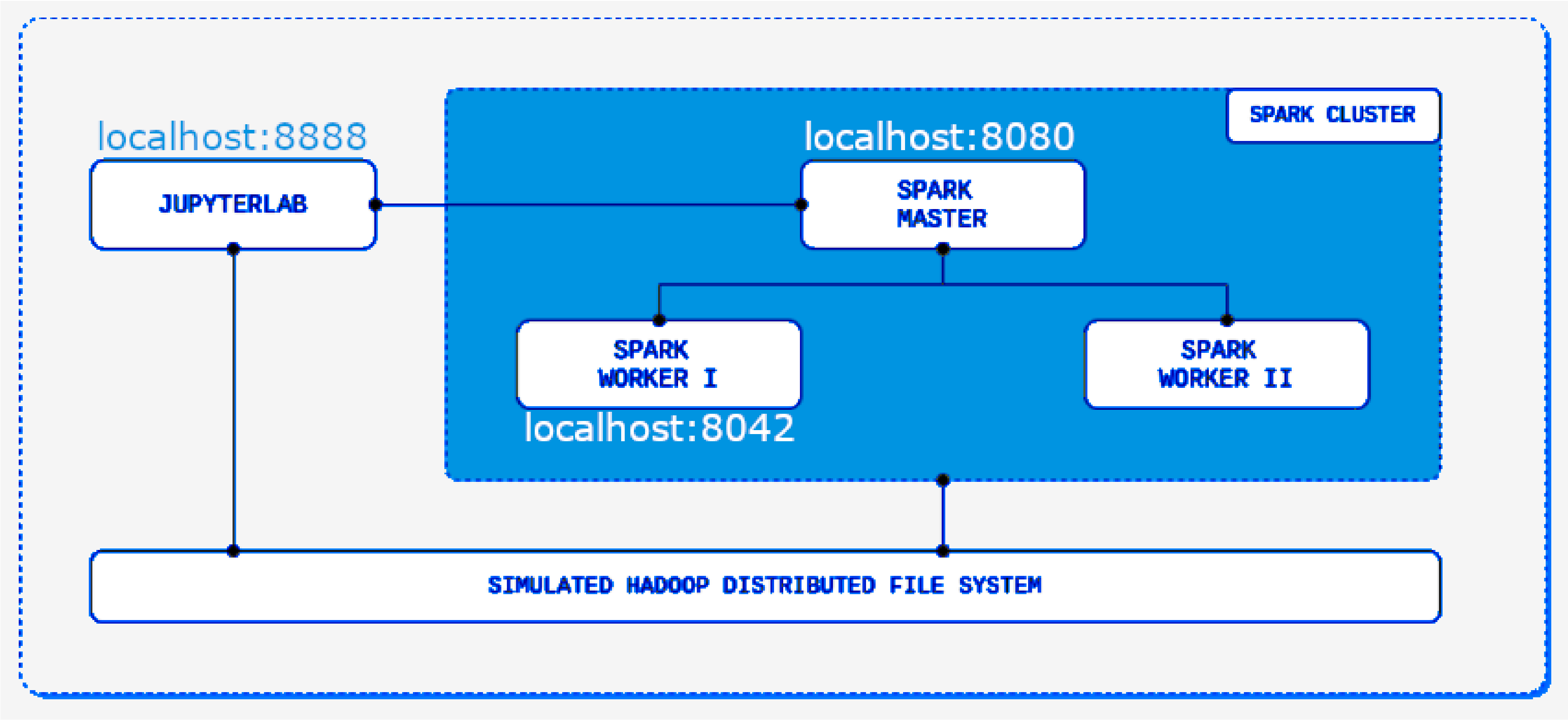
Academic year 2022-2023
May

Marco Guerra

Introduction

Cluster Architecture

- Apache Spark cluster on Docker with HDFS
- Cluster mode through Jupyter Notebook
- 2 virtual Spark workers with 5 GB memory default



DockerFile

- The Docker File has been updated from the original https://github.com/mjaglan/docker-spark-yarn-cluster-mode to support all the up to date versions of the packages:
- Hadoop: 3.3.4
- Spark: 3.3.2
- JDK: openjdk-11-jdk

Showing 3 items

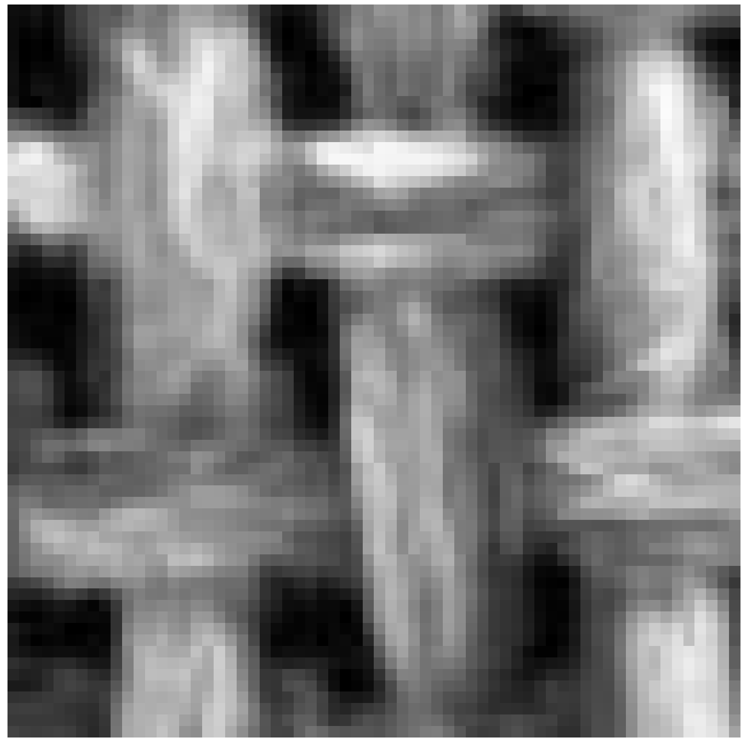
NAME	PORT(S)
testbed-master d38e8b46df7d	50070,50090,8080,8088
testbed-slave-2 3f76a50bab66	-
testbed-slave-1 5b6b9be57433	8042

Task: Binary Classification

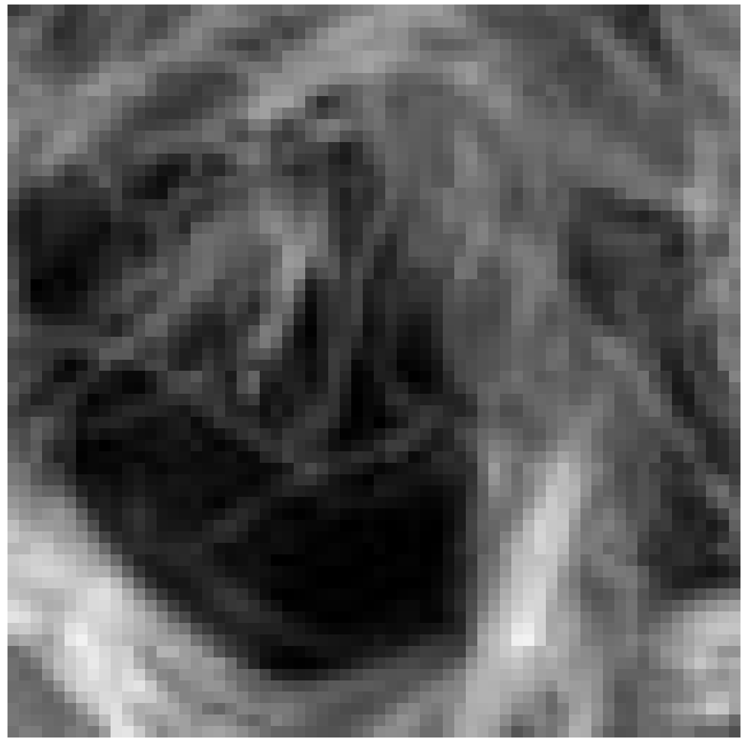
- Given a 64x64 image classify whether it contains a defect, and what kind of defect it is among 5 types.
- Five kinds of defects: color, cut, hole, thread, metal contamination

Dataset

- Dataset is from Kaggle
- Textile Defect Detection - Detection of defect in textile texture with rotations
- Built by taking 64x64 patches of high resolution images from the MVTec anomaly detection dataset(MVTec AD)
- 72.000 total samples of size 64x64, 12.000 for every category. We decide to put all defects under one class labeled 'defect'
- I used tensorflow for preprocessing



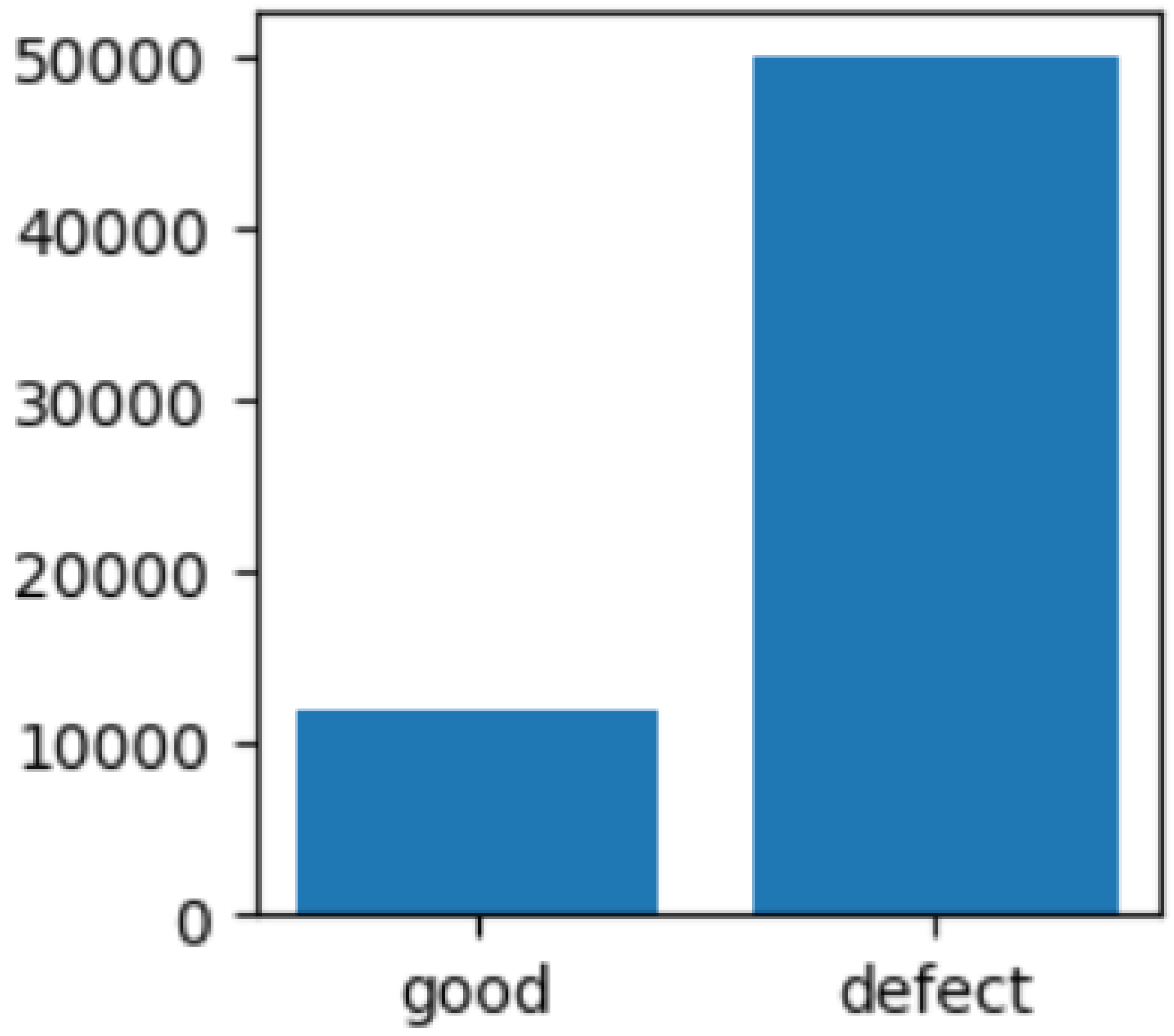
Good



Defect

Data Overview

- Unbalanced Class ratio
- Good class is only 12.000 samples while the defect class has 50.000
- This is going to be important for calculating baselines and drawing conclusions



Evaluation Metric

- For evaluating classification results we are going to use F1 score.
- Let's compute some baseline results, any improvement will be considered successful
- Random Classifier: precision simplifies to positive fraction of samples, recall simplifies to probability of classifying as positive (random is 0.5)
- Always True Classifier: precision simplifies to positive fraction of samples, recall is simply 1

Baseline Classifier	Baseline F1
Random Classifier	0.25
Always True Classifier	0.28

Features

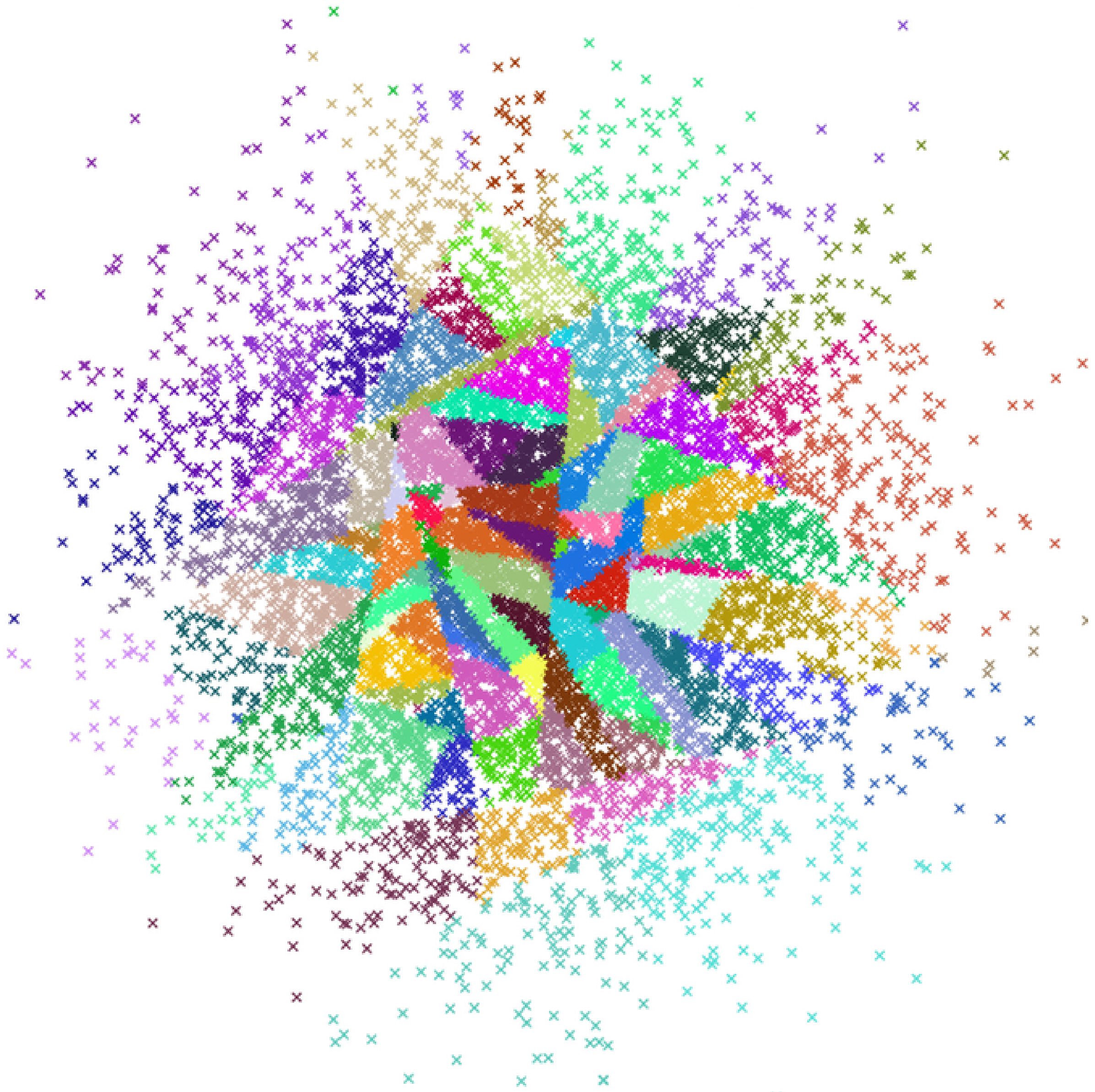
- Images are 64x64 grayscale
- These images are loaded through spark.read utility, specifying the 'image' format
- The features column is generated converting the binary data in the 'data' attribute of the image pyspark schema. An additional uniqe 'id' column is generated, required by Bucketed Random Projection
- The features dataset is split into Train and Test, respectively 60% and 40%

```
The hashed dataset where hashed values are stored in the column
+--------------------+--------------------+--------------------+
|               image|            features|                  id|
+--------------------+--------------------+--------------------+
|{file:///content/...|[130.0,130.0,130....|{file:///content/...|
|{file:///content/...|[78.0,78.0,78.0,2...|{file:///content/...|
|{file:///content/...|[187.0,187.0,187....|{file:///content/...|
|{file:///content/...|[186.0,186.0,186....|{file:///content/...|
|{file:///content/...|[130.0,130.0,130....|{file:///content/...|
|{file:///content/...|[186.0,186.0,186....|{file:///content/...|
|{file:///content/...|[130.0,130.0,130....|{file:///content/...|
|{file:///content/...|[78.0,78.0,78.0,2...|{file:///content/...|
|{file:///content/...|[187.0,187.0,187....|{file:///content/...|
|{file:///content/...|[186.0,186.0,186....|{file:///content/...|
|{file:///content/...|[166.0,166.0,166....|{file:///content/...|
|{file:///content/...|[166.0,166.0,166....|{file:///content/...|
|{file:///content/...|[127.0,127.0,127....|{file:///content/...|
|{file:///content/...|[166.0,166.0,166....|{file:///content/...|
|{file:///content/...|[152.0,152.0,152....|{file:///content/...|
|{file:///content/...|[152.0,152.0,152....|{file:///content/...|
|{file:///content/...|[15.0,15.0,15.0,2...|{file:///content/...|
|{file:///content/...|[175.0,175.0,175....|{file:///content/...|
|{file:///content/...|[32.0,32.0,32.0,2...|{file:///content/...|
|{file:///content/...|[163.0,163.0,163....|{file:///content/...|
+--------------------+--------------------+--------------------+
only showing top 20 rows
```

Modeling

Approximate Similarity Join

- Approximate similarity join takes two datasets and approximately returns pairs of rows in the datasets whose distance is smaller than a user-defined threshold
- Distances are calculated between points projections through Bucketed Random Projection, Locality Sensitive Hashing technique (bucket length = 2)
- Pros: Explainable, Easy
- Cons: Doesn't generalize



Approximate Similarity Join

- An additional 'hashes' column is creted in output to the algorithm to hold the projections
- 'image' and 'id' just differ in column name, and are replicated for simplicity's sake
- It is from the 'image' column, specifically through the 'origin' attribute that labels are derived at evaluation time

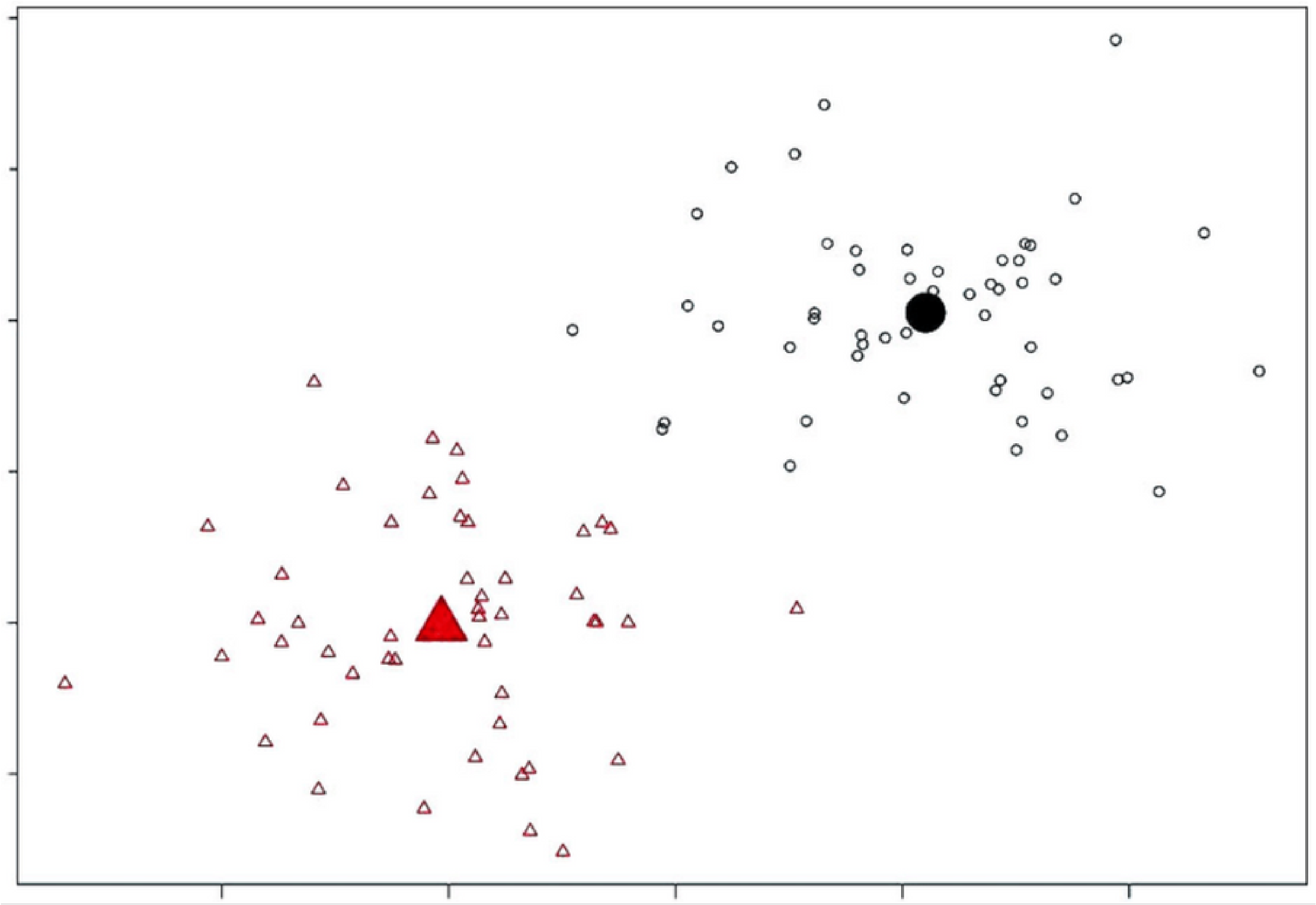
The hashed dataset where hashed values are stored in the column 'hashes':

image	features	id	LSH hashes
{file:///content/...	[130.0,130.0,130....	{file:///content/...	[[14.0], [72.0], ...
{file:///content/...	[78.0,78.0,78.0,2...	{file:///content/...	[[0.0], [26.0], [...
{file:///content/...	[187.0,187.0,187....	{file:///content/...	[[-4.0], [59.0], ...
{file:///content/...	[186.0,186.0,186....	{file:///content/...	[[-10.0], [50.0],...
{file:///content/...	[130.0,130.0,130....	{file:///content/...	[[14.0], [72.0], ...
{file:///content/...	[186.0,186.0,186....	{file:///content/...	[[-10.0], [50.0],...
{file:///content/...	[130.0,130.0,130....	{file:///content/...	[[14.0], [72.0], ...
{file:///content/...	[78.0,78.0,78.0,2...	{file:///content/...	[[0.0], [26.0], [...
{file:///content/...	[187.0,187.0,187....	{file:///content/...	[[-4.0], [59.0], ...
{file:///content/...	[186.0,186.0,186....	{file:///content/...	[[-10.0], [50.0],...
{file:///content/...	[166.0,166.0,166....	{file:///content/...	[[-21.0], [88.0],...
{file:///content/...	[166.0,166.0,166....	{file:///content/...	[[-21.0], [88.0],...
{file:///content/...	[127.0,127.0,127....	{file:///content/...	[[13.0], [60.0], ...
{file:///content/...	[166.0,166.0,166....	{file:///content/...	[[-21.0], [88.0],...
{file:///content/...	[152.0,152.0,152....	{file:///content/...	[[-10.0], [56.0],...
{file:///content/...	[152.0,152.0,152....	{file:///content/...	[[-10.0], [56.0],...
{file:///content/...	[15.0,15.0,15.0,2...	{file:///content/...	[[-9.0], [75.0], ...
{file:///content/...	[175.0,175.0,175....	{file:///content/...	[[14.0], [15.0], ...
{file:///content/...	[32.0,32.0,32.0,2...	{file:///content/...	[[-13.0], [27.0],...
{file:///content/...	[163.0,163.0,163....	{file:///content/...	[[-9.0], [77.0], ...

only showing top 20 rows

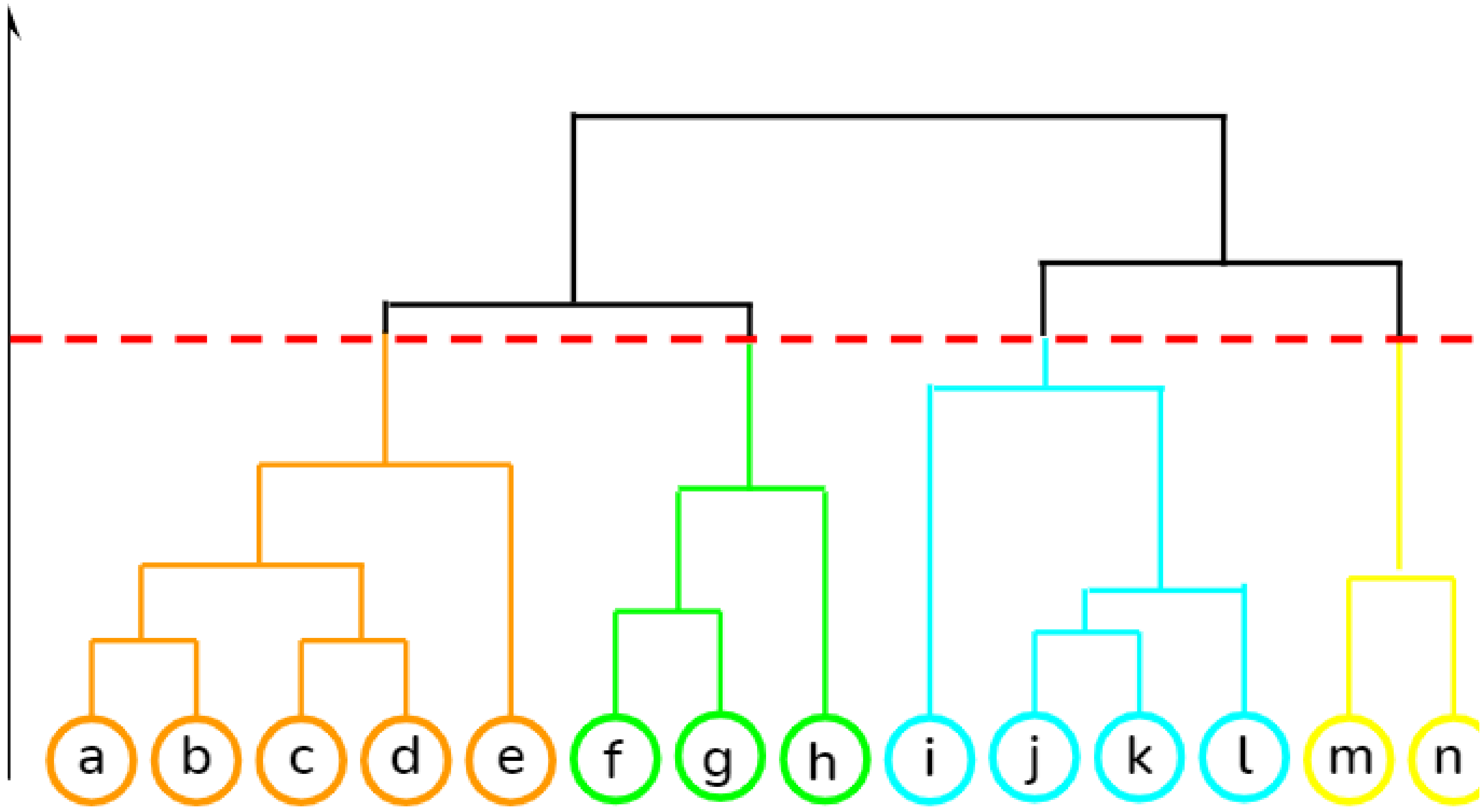
Clustering: K-Means

- Minimize within cluster variance and predict with cluster mean as centroid
- K = 2
- Decide which cluster is which class by choosing the one yielding the highest F1 score
- Pros: Generalize Better
- Cons: Unsupervised



Clustering: Bisecting K-Means

- Mixes Hierarcical and Centroid based Clustering
- K = 2
- Decide which cluster is which class by choosing the one yielding the highest F1 score
- Pros: Generalize Even Better, Faster



Results

- Results confirm what was expected
- Bisecting K-Means not only faster but also produces a different clustering
- First model is just as good as random guessing, clustering really improves by around 0.2 F1 Score points

Model	F1 Score
Approx. Similarity Join	0.23
K-Means	0.41
Bisecting K-Means	0.45