Team Research Investigation

Nikhil Keswaney, Cliffton Fernandes

December 15, 2018

Overview of the topic area

Email classification is a a binary classification technique where all the emails a user receives are classified as either *spam* or *ham*. **Spam** can be defined as unsolicited junk emails, where such emails are usually sent out in bulk by bots, illegitimate advertisers. These emails should be filtered out since such emails can contain malicious content, links to phishing websites or malware. **Ham** emails are the emails that are desired by the users and contain information beneficial to the user. For the purpose of this project, we collected the Enron corpus as out dataset which consists of approximately 500,000 emails. Since the computational size of such a dataset is too extensive to be computed easily, parallel computing can be used to reduce the computational time for such data sets.

Computational Problem Solved

Approach

The approach we used to classify the emails is the k-Nearest Neighbours (k-NN) algorithm. The k-NN algorithm is generally used in text classification and for classifying data. The k-NN algorithm works by finding the k most similar (or "closest") data points (neighbours) to the unclassified data point and then uses the labels of these points to classify the unclassified data point. k-NN finds the closeness based on a similarity coefficient.

Methodology: k-NN on Email Classification

The problem of classifying emails using the k-NN algorithm can be broken down into two parts.

1. **Training**: The training part involves processing all the available labeled data and generating a set of important attributes from the data. In our system we extract a set of words and assign a score to each of these words based on how important or how much information these words provide compared to the entire set of words from the emails.

2. **Classifying**: The words generated after the training phase are used to classify the unclassified emails. The words and their associated scores are used to find the similarity between the labeled emails and the unclassified emails. Then the k closest neighbour (most similar) emails are found using the similarity score. The email is then classified based on which category (SPAM or HAM) majority of its neighbours were labeled as.

Computations

The k-NN algorithm involves first finding the similarity score of the specific unclassified email with all the labelled emails and then finding the k closest neighbours. The similarity score is calculated by taking into account the attributes obtained from the training phase. Thus, if the number of attributes and the number of classified emails are very large, classifying an email will require a lot of computational resources and time. For example if the number of attributes are 100 and the total number of classified emails are 0.5 million, the total number of computations required would be approximately 50 million computations per unclassified email.

Parallel Approach

Parallel programming can help to reduce the time required by the above mentioned approach by performing the computations in parallel.

Analysis of the First Research Paper

- **Title**: A Novel Method for Detecting Spam Email using KNN Classification with Spearman Correlation as Distance Measure. [3]
- **Authors**: Ajay Sharma, Anil Suryawanshi.
- International Journal of Computer Applications
- **Year of Publication**: 2016
- **Link**: https://www.ijcaonline.org/archives/volume136/number6/24159-2016908471

Problem Addressed

As previously stated k-NN algorithm finds the k nearest neighbours based on a similarity score. There are multiple ways of finding the similarity score. The first paper does a comparative study of the different similarity coefficients and the effects they can have on the accuracy and speed of the classifications. This paper gave us an understanding of which similarity coefficient would work best with our system.

The similarity coefficients considered are:

Euclidean distance

Euclidean distance is a simple straight line distance if the two objects being compared have just two attributes. In case of multiple attributes the euclidean distance is the square root of the sum of the differences of all the unique attributes in the two objects being compared. In the given formula, X & Y are the values of the different attributes.

$$\boxed{dist(X,Y) = \sqrt{\sum_{i=1}^{n} \left(x_i - y_i\right)^2}}$$

The X attributes can be computed using the formula:

$$\boxed{X = \left(x_1, x_2, .., x_n\right)}$$

The Y attributes can be computed using the formula:

$$\boxed{Y = (y_1, y_2, .., y_n)}$$

Spearman's Coefficient

Spearman's coefficient gives the similarity between two objects by finding the difference in the ranks of the unique attributes of the two objects. Give a list of attributes of size n. Where x_i & y_i are the attributes of the two objects being compared.

$$\boxed{d_{ij} = 1 - \frac{6 \times \sum_{i=1}^{n} \big(rank(x_i) - rank(y_j)\big)^2}{n\big(n^2 - 1\big)}}$$

Evaluation Metrics

The paper used the following evaluation metrics to do a comparative study.

1. **Precision**

$$\boxed{P = \frac{TP}{TP + FP}}$$

2. **F-Measure**

$$\boxed{F = \frac{2 \times P \times R}{P + R}}$$

3. **Recall**

$$\boxed{R = \frac{TP}{TP + FN}}$$

4. **Accuracy**

$$\boxed{A = \frac{TP + TN}{TP + FP + FN + TN}}$$

where

1. **TP**: True Positive, that is when an email is classified as a spam and it is a spam email.
2. **TN**: True Negative, that is when an email is classified as a spam but it is not a spam.
3. **FP**: False Positive, that is when an email is classified as a spam but it is not a spam.
4. **FN**: False Negative, that is when an email is classified as a ham but it is a spam.

Results

From the results of this, paper it was clear that the Spearman's Coefficient is a better similarity coefficient when compared to the Euclidean distance. However, we eventually decided on using the Cosine Similarity Coefficient in our program. Our program was running faster with the cosine similarity coefficient as compared to the Spearman's coefficient and return results with almost the same accuracy when classifying the emails.

Analysis of the Second Research Paper

- **Title**: k-NN with TF-IDF Based Framework for Text Categorization.[4]
- **Authors**: Bruno Trstenjak, Sasa Mikac, Dzenana Donko.
- 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013
- **Year of Publication**: 2013
- **Link**: https://doi.org/10.1016/j.proeng.2014.03.129

Problem Addressed

The data available for email classifications is only text. In order to run the k-NN algorithm to classify text data, we first need to find a way to quantify this textual data. The similarity coefficient will then use these quantified values to find the nearest neighbours. The paper proposes the Term Frequency - Inverse Document Frequency concept to quantify the emails into values.

Definitions

Term Frequency - Inverse Document Frequency (TF-IDF)
TF-IDF is a numerical statistic method which allows the determination of weight for each term in each document. In simple words, it determines how important a word is in a given document as well as the entire document corpus. It is proportional to the frequency of the word in the document divided by the number of documents in the corpus. The importance of the word is increased proportionally to the number of times it appears in the documents.

Term Frequency The method computes the frequency of a particular word i in a particular document j

$$\boxed{tf(i,j) = \frac{f_{ij}}{max(f_{ij})}}$$

Inverse Term Frequency IDF method determines the relative frequency of words in a specific document through an inverse proportion of the word over the entire document corpus.

$$\boxed{idf_i = \log_2 \frac{N}{df_i}}$$

Term Frequency - Inverse Document Frequency

$$\boxed{(TF - IDF)_{ij} = \frac{f_{ij}}{max(f_{ij})} \times \log_2 \frac{N}{df_i}}$$

Results

The TF-IDF numerical statistic is a good metric to judge the importance of words in a document corpus. This score can be used to find the similarity score between two emails. Thus we have used the TF-IDF metric in the project.

Analysis of the Third Research Paper

- **Title**: Improved KNN Text Classification Algorithm with MapReduce Implementation. [5]
- **Authors**: Yan Zhao, Yun Qian, Cuixia Li.
- The 2017 4th International Conference on Systems and Informatics (ICSAI 2017)
- **Year of Publication**: 2017
- **Link**: https://ieeexplore.ieee.org/document/8248509

Problems Addressed

k-NN is a widely used classification algorithm but when dealing with large amounts of data the sequential program would be inadequate. This paper proposes the idea of implementing the K-NN algorithm as a map reduce job so that it can be used to process large amount of data.

Implementation in MAP-REDUCE.

- The basic idea of k-NN as a MAP-Reduce job.
  - Calculate the (TF-IDF-DF) score matrix according to the formulae.
  - Obtain Feature Vectors.
  - Vectorize training sets.
- **MAP** : Calculate the distance of that vector with all the training data vectors.
- **REDUCE**: Merge the results of the MAP outputs find the K nearest neighbours and assign the category that it has maximum neighbours with.

Results

The idea proposed by the paper to parallelize the k-NN algorithm formed a reference for our eventual implementation of the parallel program for email classification.

Sequential Program

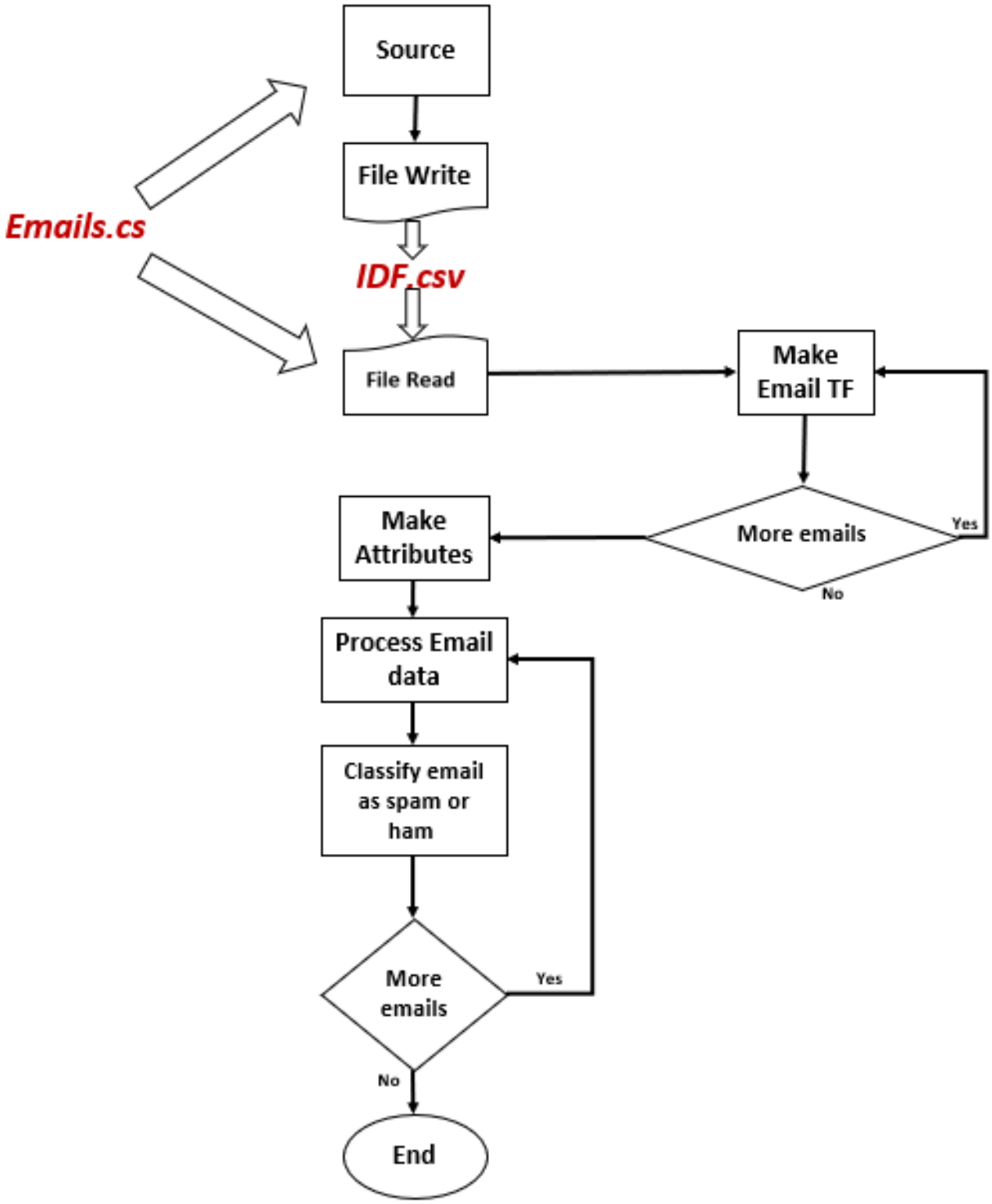

Figure 1: Sequential Program

The program has 2 parts, the first generates the IDF words and their associated scores. The second part of the program reads the file generated from the first program as an input. It is also responsible for finding the best words required for computing the similarity and classifying the emails from the unclassified list. After classifying all the emails it saves the result into an output file.

Algorithm

Part 1

- The first step of the program loads the data from the input files (Classified as well as unclassified).
- The second step of the program generates the TF-IDF scores and saves it to a file.

Part 2

- The first step of the program reads the file generated by the first program.
- The second step of the program selects the words with the highest TF-IDF score.
- The next phase of the system begins the classification part. It repeats for all the unclassified emails.
  - Iterate over the classified emails and find the similarity scores to the unclassified email.
  - Find the k nearest neighbours of the unclassified email.
  - Find the majority category of the k nearest neighbours.
  - Classify the email according to the majority vote.
- The last step of the system saves all the newly classified emails into an output file.

Parallel Program

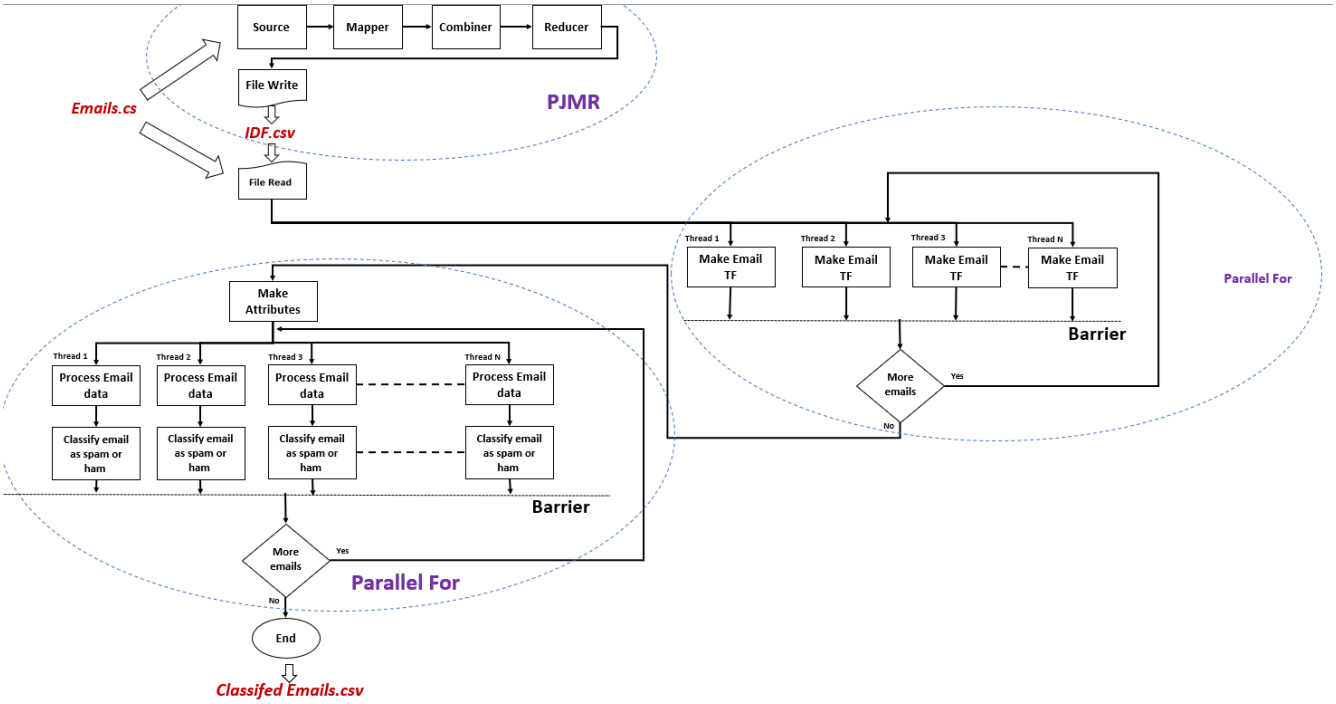

Figure 2: Parallel Program

The parallel program has two parts the first part is a Parallel Java Map Reduce Program that generates the IDF words and their associated scores and saves it as a *csv* file. The second part of the program is the part that classifies emails. The second part is further divided into two sections.

1. The first section iterates over all the emails in parallel and calculates the TF scores.
2. The second section finds the words with the highest TF-IDF scores, this part cannot be parallelized as it is has a sequential dependency. After finding the best words the program iterates over the unclassified emails in parallel and performs classification by running the k-NN algorithm.

Algorithm

Part 1

Reads the input files and generates the IDF scores in parallel.

- The first step of the program loads the data from the input files (Classified as well as unclassified).
- The second step of the program generates the TF-IDF scores by iterating over all the emails in parallel.
- The results are saved to a csv file.

Part 2: Section 1

- The first step of the program reads the file generated by part 1.

- The program iterates over all the emails and calculates the TF scores in parallel.
- This data is provided to the next section of the program.

Part 2: Section 2

- The first step is to find the words with the highest TF-IDF scores this is sequential.
- The next phase of the system begins the classification part. It repeats for all the unclassified emails and is done in parallel.
  - Iterate over the classified emails and find the similarity scores to the unclassified email.
  - Find the k nearest neighbours of the unclassified email.
  - Find the majority category of the k nearest neighbours.
  - Classify the email according to the majority vote.
- The last step of the system saves all the newly classified emails into an output file.

Developer's Manual

Requirements

1. Java 1.7
2. Parallel Java 2 Library

Compilation

1. Export the java and javac binaries to your path.
2. Export the pj2.jar library to your classpath: *export CLASSPATH=.:<PATH_TO_PJ2_JAR>*
3. Unzip the zip file. *unzip TeamRavana.zip*
4. Change directory into the main source folder. *cd TeamRavana/src*
5. Compile the source code. *javac *.java*
6. Creating the Jar file. *jar -cf proj.jar *.class*

A user's manual

Program 1

- Input: There are 3 CSV files that needs to be given to the first program The input will be the complete email data i.e. classified spam, classified ham, unclassified. Each of these CSV's will have the same format.
  1st value can be any value we have selected an index .
  The 2nd value will be the category of the email, which will be as follows.

1. 1 - SPAM
2. 0 - HAM
3. 2 - UNCLASSIFIED.

The 3rd value will be the actual text of the email.
ALL SEPARATED BY COMMA'S (,)
Here is an example.

```
1,0, subject neon for march 28 here is the neon lesson for march 28 th   experiencing god week 4  doc
2,0, subject need deal for march 2000 daren   sherlyn schumack suggested that i contact you with this matter  i am working on some old hpl issues and
have come across a deal that was initially set up    208101   but this deal was subsequently killed  no reason found   this deal was copied from
205533 which was good through 2  29  00  please set up a valid deal for march 2000 just like either one of these deals whereby hpl is buying gas from
ena on tejas at the same points  this deal is needed because it leads to a valid sale to reliant energy  please note that reliant has been invoiced
and settlement has already been made  if i have not provided enough information or if you have any questions   please do not hesitate to give me a
call  as i need to resolve this issue as soon as possible   your prompt attention to this matter would be most appreciated  buddy x  31933
3,0, subject neon lesson  5 please respond to here is the lesson for lesson 5  have fun   experiencing god week 5  doc
```

Figure 3: Strong scaling table

- Running the Sequential program we will use the following command.
  usage: java pj2 jar=<jar> MakeIdfScoreSeq <HAM> <SPAM> <UNCLASSIFIED> <IDF>
  a. <HAM> The location of the classified Ham file.
  b. <SPAM> The location of the classified SPAM file.
  c. <UNCLASSIFIED> The location of the unclassified file.
  similarity. <IDF> The location of the IDF file where the results should be stored.

  Running the Parallel program we will use the following command.
  usage: java pj2 jar=<jar> threads=<NT> MakeIdfScoreSmp <HAM> <SPAM> <UNCLASSIFIED> <IDF>
  a. <HAM> The location of the classified Ham file.
  b. <SPAM> The location of the classified SPAM file.
  c. <UNCLASSIFIED> The location of the unclassified file.
  similarity. <IDF> The location of the IDF file where the results should be stored.

- Output File: The output file will also be a CSV where the
  1st element will be the word 2nd element will be its corresponding IDF score. ALL SEPARATED BY COMMA'S (,)

Program 2

- Input: There are 4 CSV files that needs to be given to the second program The 1st 3 input files will be all the email data i.e. classified spam, classified ham, unclassified. Each of these CSV's will have the same format. The 4th file will be the output of the 1st program Following is how the 1st 3 input files should be like:-

1st value can be any value we have selected an index .
2nd value will be the category of the email. which will be as follows.

1. 1 - SPAM
2. 0 - HAM
3. 2 - UNCLASSIFIED.

The 3rd value will be the actual text of the email.
ALL SEPARATED BY COMMA'S (,)
Here is an example.

```
1,0, subject neon for march 28 here is the neon lesson for march 28 th   experiencing god week 4  doc
2,0, subject need deal for march 2000 daren   sherlyn schumack suggested that i contact you with this matter  i am working on some old hpl issues and
have come across a deal that was initially set up    208101   but this deal was subsequently killed  no reason found   this deal was copied from
205533 which was good through 2  29  00  please set up a valid deal for march 2000 just like either one of these deals whereby hpl is buying gas from
ena on tejas at the same points  this deal is needed because it leads to a valid sale to reliant energy  please note that reliant has been invoiced
and settlement has already been made  if i have not provided enough information or if you have any questions   please do not hesitate to give me a
call  as i need to resolve this issue as soon as possible   your prompt attention to this matter would be most appreciated  buddy x  31933
3,0, subject neon lesson  5 please respond to here is the lesson for lesson 5  have fun   experiencing god week 5  doc
```

Figure 4: Strong scaling table

THE 4TH FILE WHICH IS THE OUTPUT OF THE 1ST PROGRAM SHOULD NOT BE CHANGED AND DIRECTLY GIVEN A INPUT TO THIS PROGRAM.

- Running the Sequential program we will use the following command.
  usage: java pj2 jar=<jar> EmailClassifierSeq <HAM> <SPAM> <UNCLASSIFIED> <IDF> <CLASSIFIED>
  a. <HAM> The location of the classified Ham file.
  b. <SPAM> The location of the classified SPAM file.
  c. <UNCLASSIFIED> The location of the unclassified file.
  similarity. <IDF> The location of the IDF file where the results of the computed IDF's are stored.
  d. <CLASSIFIED> This is an optional input if you want the classified data to go in a particular directory

  Running the Parallel program we will use the following command.
  usage: java pj2 jar=<jar> threads=<NT> MakeIdfScoreSmp <HAM> <SPAM> <UNCLASSIFIED> <IDF>
  a. <HAM> The location of the classified Ham file.
  b. <SPAM> The location of the classified SPAM file.
  c. <UNCLASSIFIED> The location of the unclassified file.
  similarity. <IDF> The location of the IDF file where the results should be stored.

- Output File: The output file will also be a CSV where the
  1st element will be the word 2nd element will be its corresponding IDF score. ALL SEPARATED BY COMMA'S (,)

The strong scaling performance data

Running Part 1

We ran our first program on the following data sets.

1. Unclassified dataset of 50 emails.
2. Unclassified dataset of 100 emails.
3. Unclassified dataset of 200 emails.
4. Unclassified dataset of 500 emails.
5. Unclassified dataset of 800 emails.
6. Unclassified dataset of 4200 emails.
7. Unclassified dataset of 10000 emails.
8. Unclassified dataset of 100000 emails.

on threads 1, 2, 4, 8, 10, 12.

Speed up						
	1	2	4	8	10	12
50	1	0.6	0.5	0.5	1	0.5
100	1	0.6	0.7	0.6	0.1	0.9
200	1	0.5	0.5	0.6	0.7	1
500	1	0.7	0.7	0.7	1	1
800	1	0.6	0.6	0.7	0.6	0.6
4200	1	1	1	1	1	1
10000	1	1	1	1	1	1
100000	1	1	1	0.9	0.9	0.8

Figure 5: Strong scaling table

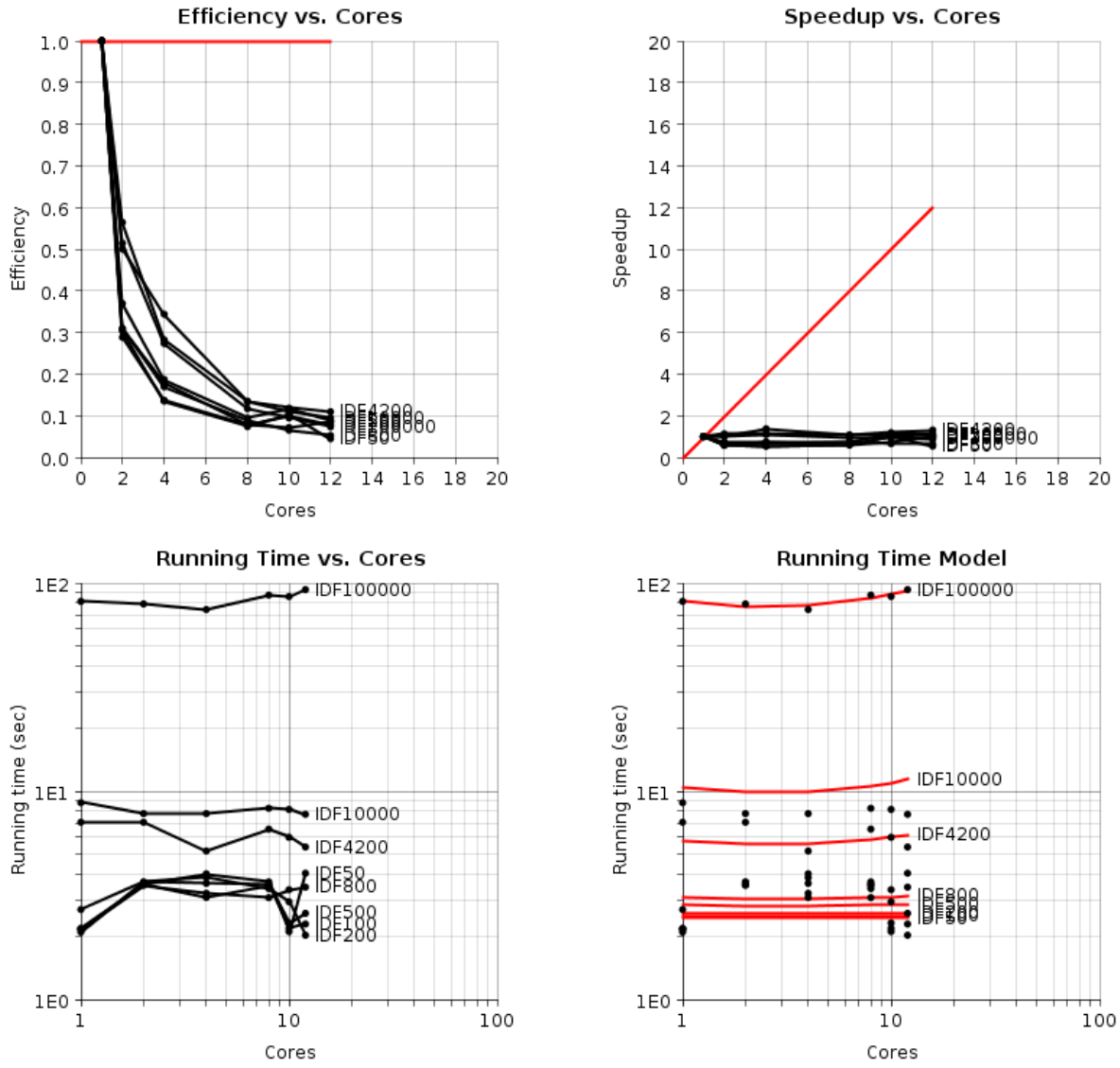

Figure 6: Strong Scaling

This program showed very bad strong scaling as we can see which will be explained in the bellow section

Running Part 2

We ran our second program for the following data sets:

1. Unclassified dataset of 50 emails.
2. Unclassified dataset of 100 emails.
3. Unclassified dataset of 200 emails.
4. Unclassified dataset of 500 emails.
5. Unclassified dataset of 800 emails.

And we ran it on all threads from 0 to 12. This is what we got as the results got on our second program (where email classification is performed)

DataSets vs Cores

Data File	1	2	3	4	5	6	7	8	9	10	11	12
unclassified50	151716	80751	52222	39562	32122	29057	25804	22693	20870	17028	16531	15972
unclassified100	298757	146340	97823	72162	59168	53043	43085	39121	35160	29604	29545	26826
unclassified200	616551	294576	206142	147938	117154	104416	91074	80832	71524	60640	61213	62916
unclassified500	1403556	707908	493743	381581	280563	253024	214524	176320	170264	139991	130302	126359
unclassified800	2286755	1156601	763830	580852	460445	411131	351104	289402	271994	230785	214004	211912

Efficiency

Data File	Core 1	Cores 2	Cores 3	Cores 4	Cores 5	Cores 6	Cores 7	Cores 8	Cores 9	Cores 10	Cores 11	Cores 12
unclassified50	1	0.9394063231	0.9684041209	0.958723017	0.9446236224	0.8702206009	0.8399362225	0.8356982329	0.8077303945	0.8909795631	0.8343332912	0.7915727523
unclassified100	1	1.020763291	1.018013939	1.035021895	1.009860059	0.9387258136	0.9905900297	0.9545928018	0.9441189483	1.009177814	0.9192664503	0.9280704043
unclassified200	1	1.046505825	0.9969681094	1.041907759	1.052547928	0.984125996	0.9671115169	0.9534451084	0.9577997129	1.016739776	0.9156566564	0.8166324941
unclassified500	1	0.9913406827	0.9475617882	0.9195662258	1.000528224	0.9245209941	0.9346646529	0.9950345962	0.9159344704	1.002604453	0.9792328591	0.9256404372
unclassified800	1	0.9885669302	0.9979336589	0.9842244668	0.9932804135	0.9270179902	0.9304345314	0.9877069785	0.9341525508	0.9908594579	0.9714155725	0.899254958

Speed Up

Data File	Core 1	Cores 2	Cores 3	Cores 4	Cores 5	Cores 6	Cores 7	Cores 8	Cores 9	Cores 10	Cores 11	Cores 12
unclassified50	1	1.878812646	2.905212363	3.834892068	4.723118112	5.221323605	5.879553558	6.685585863	7.269573551	8.909795631	9.177666203	9.498873028
unclassified100	1	2.041526582	3.054056817	4.140087581	5.049300297	5.632354882	6.934130208	7.636742415	8.497070535	10.09177814	10.11193095	11.13684485
unclassified200	1	2.093011651	2.990904328	4.167631035	5.262739642	5.904755976	6.769780618	7.627560867	8.620197416	10.16739776	10.07222322	9.799589929
unclassified500	1	1.982681365	2.842685365	3.678264903	5.002641118	5.547125964	6.54265257	7.96027677	8.243410234	10.02604453	10.77156145	11.10768525
unclassified800	1	1.97713386	2.993800977	3.936897867	4.966402068	5.562107941	6.51304172	7.901655828	8.407372957	9.908594579	10.6855713	10.7910595

Figure 7: strong scaling table

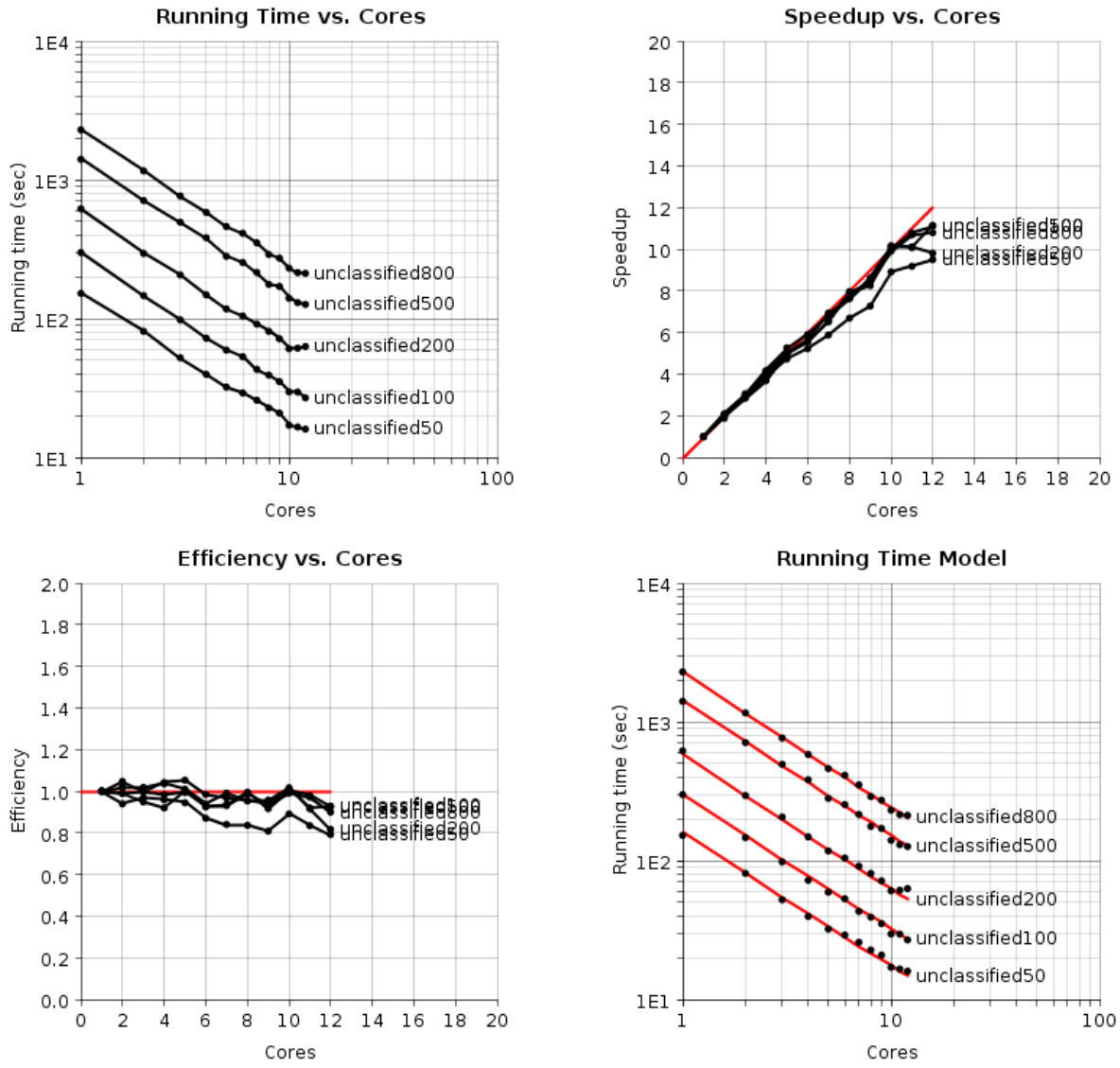

Figure 8: Strong Scaling

We got a close to ideal scaling when we did strong scaling. But there is a small drop in the speed-up and efficiency graph as the number of cores increases which will be explained in the next section.

An explanation of why any non-ideal strong scaling is occurring

Running Part 1

The reason we got bad scaling for the first part of the program is because we are using small datasets on huge resources we implemented the parallel program for calculating the document frequency (df) using the pjmr library which uses a cluster to do map reduce for an example of around 50 data sets the pjmr is using a lot of resources and has a lot of communication overhead as it is a cluster. which increases the running time of the program even more than the sequential program.

Running Part 2

There is a drop in the graph for smaller data sets when we increase the number of cores. This is because when we increase the number of cores for the same dataset the parallelizable part of the program becomes smaller and smaller but the sequential part i.e. file I/O, sorting remains constant due to which we don't see the expected increase in the speed-up graph.

The weak scaling performance data

Running Part 1

We ran our first program on the following way.

1. Unclassified dataset of size 50.
2. Unclassified dataset of size 100.
3. Unclassified dataset of size 150.
4. Unclassified dataset of size 200.
5. Unclassified dataset of size 300.

Each of these dataset was ran on 1, 2, 4, 8, 12 threads. These are the results we got.

n(1)	K	n(K)	T(1)	T(k)	Size up	Efficencey
300	1	300	881735	881735	1	1
	2	600	881735	848595	2.07	1.03
	4	1200	881735	849327	4.15	1.03
	10	3000	881735	937783	9.4	1
	12	3600	881735	1015807	10.41	1
50	1	50	163370	163370	1	1
	2	100	163370	141666	2.3	1.1
	4	200	163370	150893	4.33	1.08
	10	500	163370	142763	11.4	1.14
	12	600	163370	160377	12.22	1.01
100	1	100	301177	301177	1	1
	2	200	301177	291971	206	1.03
	4	400	301177	297213	4.05	1.01
	10	1000	301177	289736	10.39	1.03
	12	1200	301177	308833	11.7	0.97
150	1	150	469327	469327	1	1
	2	300	469327	438757	2.1	1.06
	4	600	469327	424377	4.4	1.1
	10	1500	469327	425454	11.03	1.1
	12	1800	469327	472085	11.9	0.99
200	1	200	584284	584284	1	1
	2	400	584284	590564	1.97	0.99
	4	800	584284	574599	4.06	1.01
	10	2000	584284	569985	10.25	1.02
	12	2400	584284	671459	10.44	0.87

Figure 9: Strong scaling table

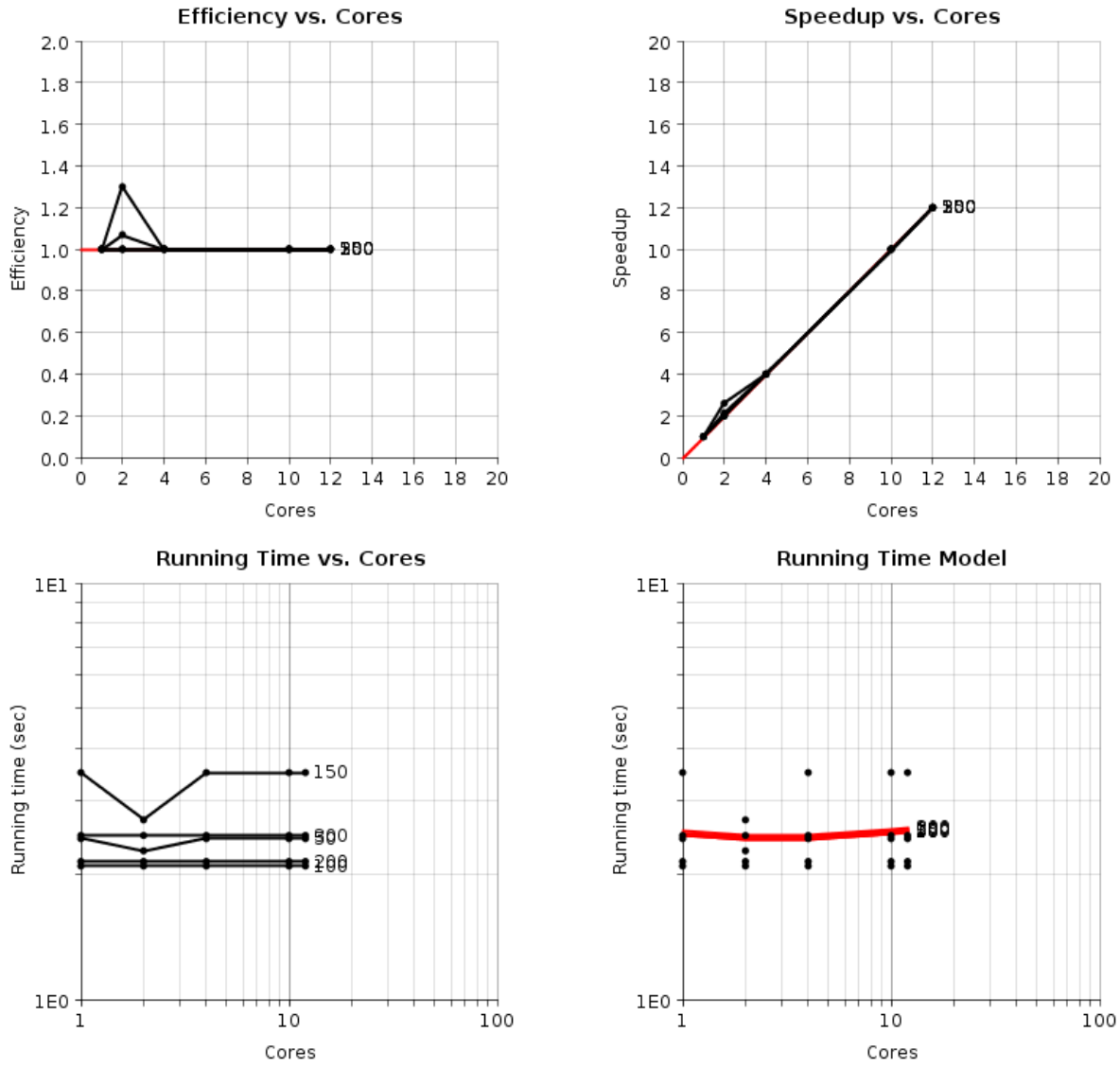

Figure 10: Strong Scaling Graphs

For weak scaling we got an ideal scaling and there is no drop in the running time of the program also.

Running Part 2

We ran our second program for the following data sets:

1. Unclassified dataset of size 50.
2. Unclassified dataset of size 100.
3. Unclassified dataset of size 150.
4. Unclassified dataset of size 200.
5. Unclassified dataset of size 300.

Each of these dataset was ran on 1, 2, 4, 8, 12 threads. These are the results we got.

n(1)	K	n(K)	T(1)	T(k)	Size up	Efficencey
300	1	300	881735	881735	1	1
	2	600	881735	848595	2.07	1.03
	4	1200	881735	849327	4.15	1.03
	10	3000	881735	937783	9.4	1
	12	3600	881735	1015807	10.41	1
50	1	50	163370	163370	1	1
	2	100	163370	141666	2.3	1.1
	4	200	163370	150893	4.33	1.08
	10	500	163370	142763	11.4	1.14
	12	600	163370	160377	12.22	1.01
100	1	100	301177	301177	1	1
	2	200	301177	291971	206	1.03
	4	400	301177	297213	4.05	1.01
	10	1000	301177	289736	10.39	1.03
	12	1200	301177	308833	11.7	0.97
150	1	150	469327	469327	1	1
	2	300	469327	438757	2.1	1.06
	4	600	469327	424377	4.4	1.1
	10	1500	469327	425454	11.03	1.1
	12	1800	469327	472085	11.9	0.99
200	1	200	584284	584284	1	1
	2	400	584284	590564	1.97	0.99
	4	800	584284	574599	4.06	1.01
	10	2000	584284	569985	10.25	1.02
	12	2400	584284	671459	10.44	0.87

Figure 11: Weak scaling table

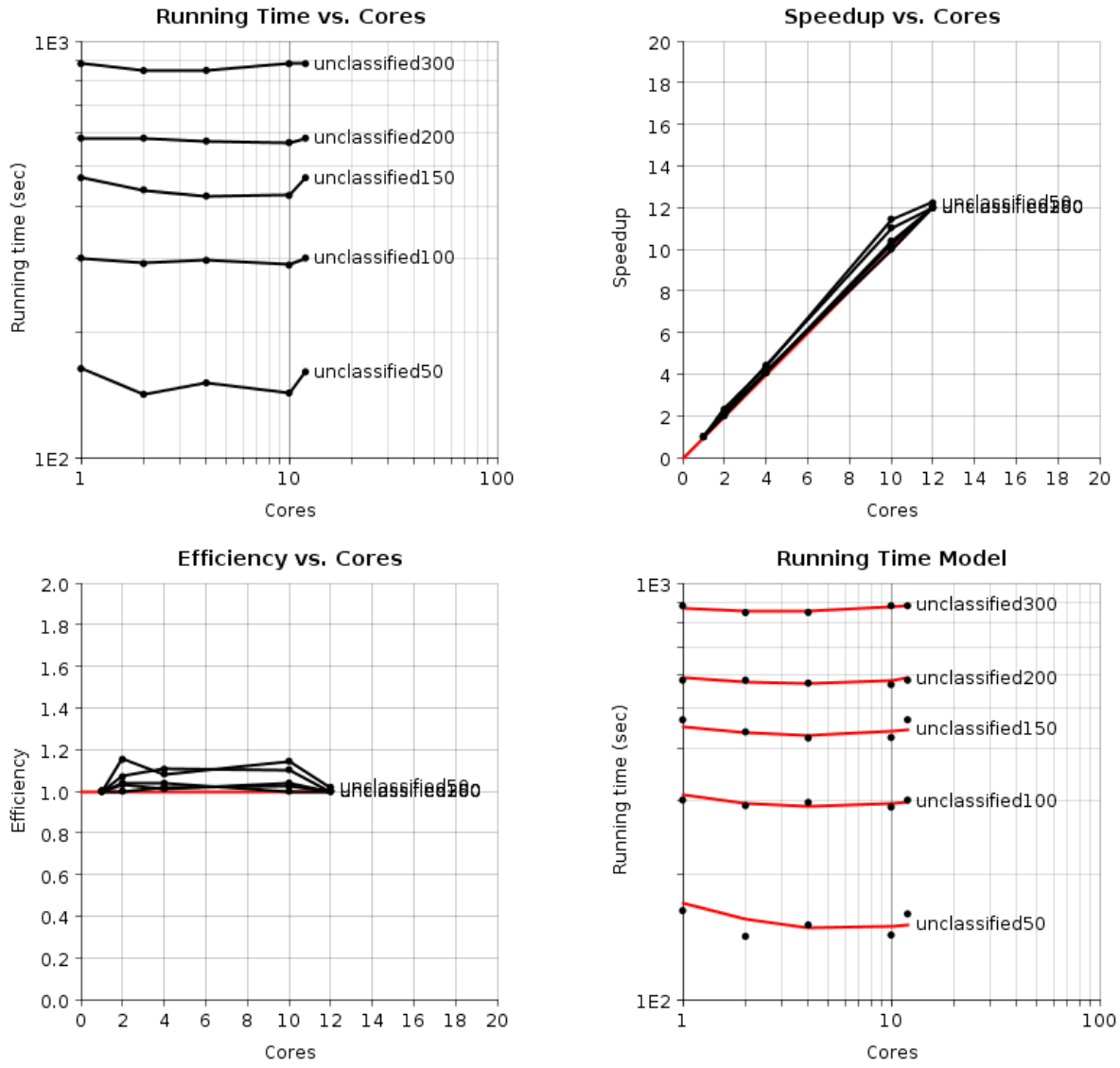

Figure 12: Weak Scaling

We got a better than ideal scaling but here also there is a drop in efficiency and the speedup graph which is explained in the next section.

An explanation of why any non-ideal weak scaling is occurring

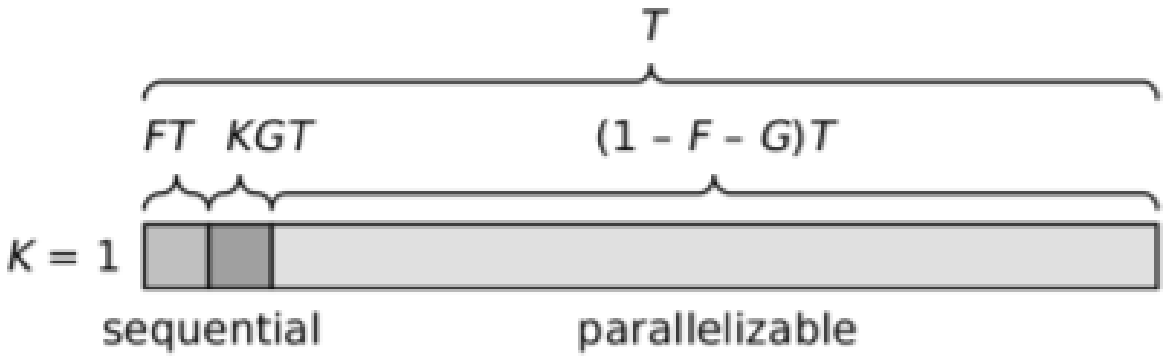

Figure 13: Division according to time

In the above figure we can see that there are 3 parts in the program.

1. F - Which is the sequential part which will not change even though the dataset size increases.
2. KG - Which is the sequential part which will increase as the dataset size increases.
3. (1-F-G) - Which is the parallelizable part.

F - fraction of our program is the selecting of words and sorting part as we always select number of words that depends upon the classified part and the I/O part of the classified part. G - Of our program will be the I/O for the unclassified part.

So as we are increasing the dataset size the size of G is also increasing that is the reason its showing a drop as we increase the number of cores.

A discussion of possible future work

Hereafter, the project can be extended by converting our current implementation into a cluster parallel program which can be run on multiple nodes such that every chunk of unclassified set of emails can be executed on each node, with the similarity score being measured by the individual threads. This enhancement can improve the computational time, thus enabling data sets of much larger sizes to be classified in a reduced span of time.

A discussion of what you learned from the project

By implementing this project, some of the major concepts we learned are how to perform implementation of k Nearest Neighbours in parallel, impact of sequential dependencies and how to handle them. Also, the impact of data structure selection on the running time and the impact I/O has on running time were some other concepts we were exposed to.

A statement of what each individual team member did on the project

- **Cliffton Fernandes**: Worked on the classification parts of the system. i.e part 2 of the seqencial and the parallel algorithms.
- **Nikhil Keswaney**: Worked on the parts of the system that generated the TF-IDF scores. i.e part 1 of the sequential and the parallel algorithms.
- Both the team members contributed to the research work, presentations, reports equally. Usually working on different sections of the tasks before aggregating the results.

References

[1] Enron Corpus, Retrieved from https://www.cs.cmu.edu/ ./enron/ Enron corpus dataset

[2] Parallel Java 2 Library, Retrieved from https://www.cs.rit.edu/ ark/pj2.shtml

[3] Ajay Sharma, Anil Suryawanshi *A Novel Method for Detecting Spam Email using k-NN Classification with Spearman Correlation as Distance Measure.* International Journal of Computer Applications, Year of Publication: 2016, Link: https://www.ijcaonline.org/archives/volume136/number6/24159-2016908471

[4] Bruno Trstenjak, Sasa Mikac, Dzenana Donko. *k-NN with TF-IDF Based Framework for Text Categorization.* 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013, Year of Publication: 2013 , Link:https://doi.org/10.1016/j.proeng.2014.03.129

[5] Yan Zhao, Yun Qian, Cuixia Li *Improved k-NN Text Classification Algorithm with MapReduce Implementation* The 2017 4th International Conference on Systems and Informatics (ICSAI 2017) Year of Publication: 2017, Link:https://ieeexplore.ieee.org/document/8248509

[6] Alan Kaminsky, *BIG CPU, BIG DATA: Solving the World's Toughest Computational Problems with Parallel Computing.*, 2016