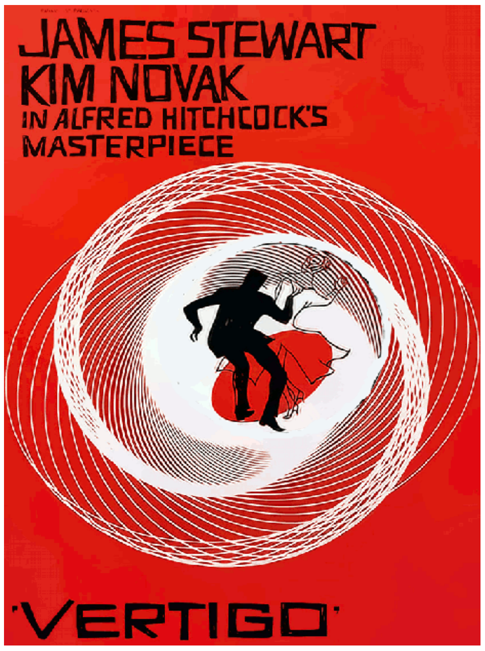
SENTIMENT ANALYSIS WITH NAIVE BAYES CLASSIFIER ON A DISTRIBUTED NETWORK

KAUSHIK VARANASI 201201111
PRERNA GAUTAM 201201110

snatch
MICKEY
TYRONE
BULLET-TOOTH TONY
BRICK TOP
TURKISH
TOMMY
BENICIO DEL TORO DENNIS FARINA VINNIE JONES
BRAD PITT RADE SHERBEDGIA JASON STATHAM
"A rock-'em sock-'em caper with energy to spare."



PROJECT DESCRIPTION

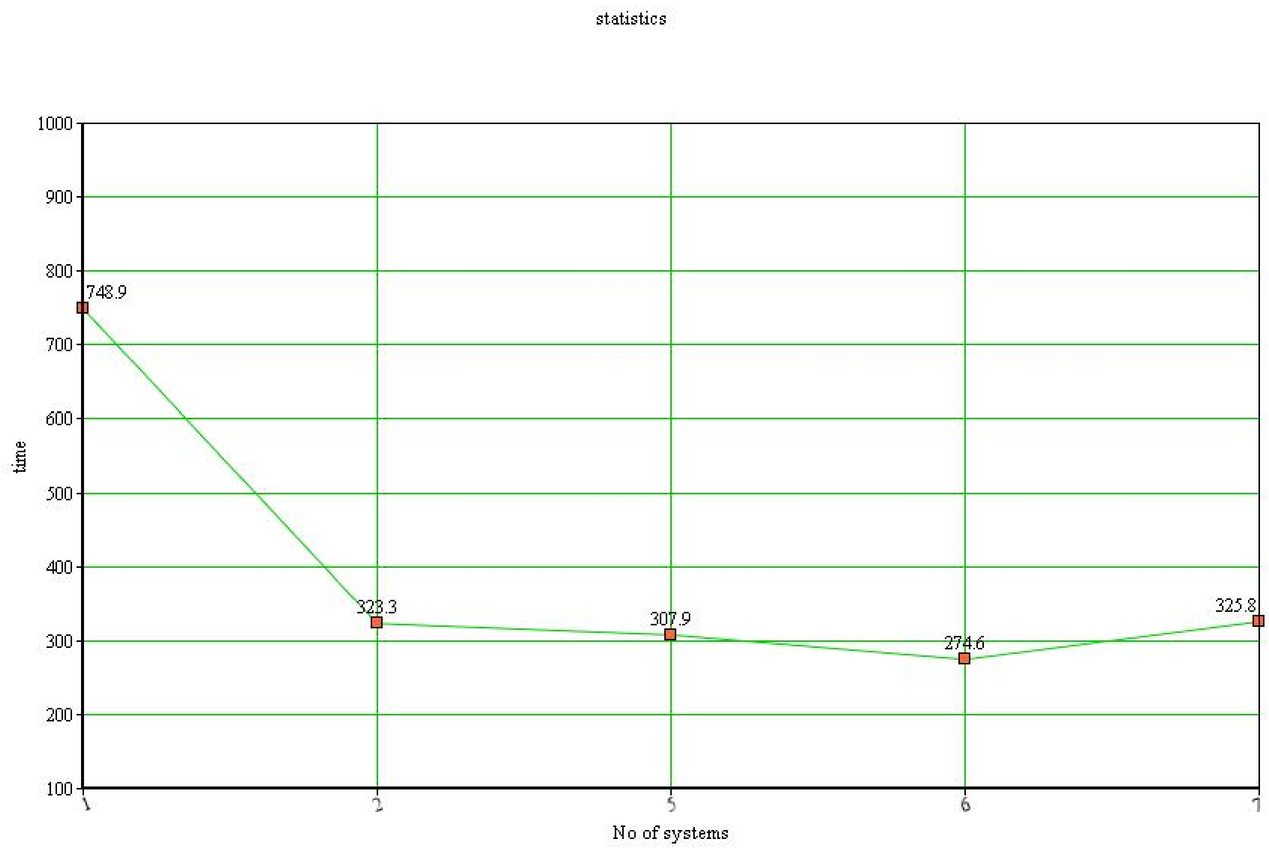
The idea is to classify a movie as good or bad by analysing keywords in the movie review, provided as text file and then passing them through an already trained NAIVE BAYES CLASSIFIER.
This is implemented over a distributed network to implement fast I/O methods described in the benchmarks.

The project had various number of challenges as it is not a trivial one and had to be hard coded from scratch:

- implementing a distributed system, prototyping a client-server model, and then running benchmark tests over simple tasks like finding word-frequencies

- parsing the large data obtained from cornell university into readable and passable format to the classifiers
- simulating the clients and making them synchronously work with the server
- making installers and bash scripts for the software

BENCHMARKING



WORKING

Basically the classifier is run on the machine which has to be fed with about 10 MB of data. It calls a Server() object which interacts with N clients(passed as an argument). Clients are generated with a bash script. The movie review files are shared across all the clients. The following things happen:

1. The classifier object is run, which calls a server object.
2. The server is initiated and waits for connection
3. N clients are simulated on different terminals each ready to take the job
4. once these connections are made, the time is noted for benchmarking.
5. each client is assigned approximately T/N files to read and parse from.

6. Each client reads from its set of files simultaneously and sends the parsed data to the server. This is nothing but a dictionary of objects and their occurrences.
7. the server receives this data and packs it and sends to the classifier for training.
8. A test data is randomly generated and paSSED to the classifier and results are calculated.

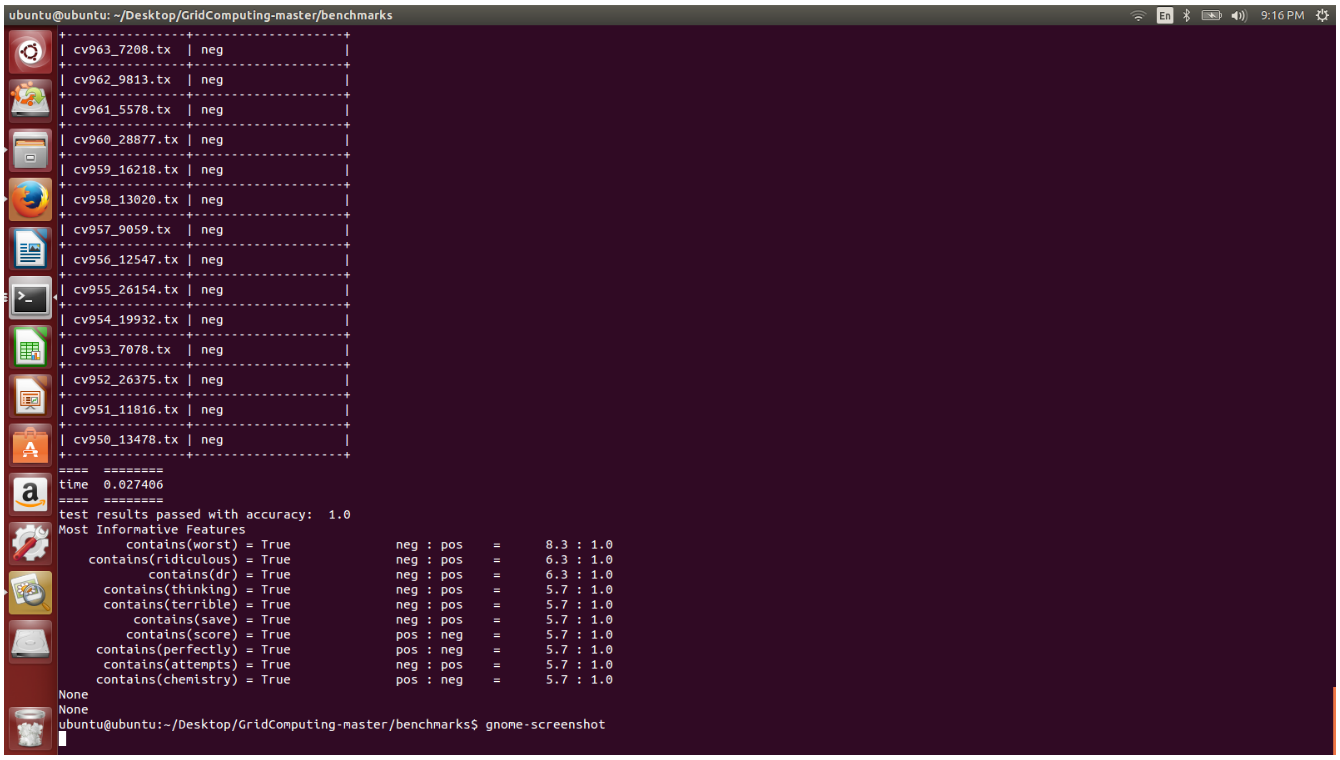
The system resources and time consumed is measured for benchmarking and a performance graph is plotted.

USAGE

# bash install.sh
# python classifier.py <number of slaves to create>

on each slave terminal run this (will create a script that does this automatically soon !!)

# python client.py



```
+----------------+-------------------+
| cv963_7208.tx  | neg               |
+----------------+-------------------+
| cv962_9813.tx  | neg               |
+----------------+-------------------+
| cv961_5578.tx  | neg               |
+----------------+-------------------+
| cv960_28877.tx | neg               |
+----------------+-------------------+
| cv959_16218.tx | neg               |
+----------------+-------------------+
| cv958_13020.tx | neg               |
+----------------+-------------------+
| cv957_9059.tx  | neg               |
+----------------+-------------------+
| cv956_12547.tx | neg               |
+----------------+-------------------+
| cv955_26154.tx | neg               |
+----------------+-------------------+
| cv954_19932.tx | neg               |
+----------------+-------------------+
| cv953_7078.tx  | neg               |
+----------------+-------------------+
| cv952_26375.tx | neg               |
+----------------+-------------------+
| cv951_11816.tx | neg               |
+----------------+-------------------+
| cv950_13478.tx | neg               |
+----------------+-------------------+
====  ========
time  0.027406
====  ========
test results passed with accuracy:  1.0
Most Informative Features
         contains(worst) = True              neg : pos    =      8.3 : 1.0
    contains(ridiculous) = True              neg : pos    =      6.3 : 1.0
            contains(dr) = True              neg : pos    =      6.3 : 1.0
      contains(thinking) = True              neg : pos    =      5.7 : 1.0
      contains(terrible) = True              neg : pos    =      5.7 : 1.0
          contains(save) = True              neg : pos    =      5.7 : 1.0
         contains(score) = True              pos : neg    =      5.7 : 1.0
     contains(perfectly) = True              pos : neg    =      5.7 : 1.0
      contains(attempts) = True              neg : pos    =      5.7 : 1.0
     contains(chemistry) = True              pos : neg    =      5.7 : 1.0
None
None
ubuntu@ubuntu:~/Desktop/GridComputing-master/benchmarks$ gnome-screenshot
```

RESULTS

Results were not so great but good enough for a workaround and the minimal effort. As seen in the benchmark graph, the values started dropping at cpus=7. **SO WE COULD SUCCESSFULLY SCALE IT UP TO 6 COMPUTERS.**

PROJECT APPLICATION

This is used to know more about the sentiment analysis of various movies by scraping data related to them from social networks like Twitter, Facebook etc. This intern provides a valuable feedback to the movie industry.
This is not just limited to movie reviews. The core of this project is making a scalable distributed network, which can be used on a range of application involving big data.

FUTURE PROSPECTS

The future prospects would be to do a more complicated analysis(like protein structure analysis) over a more scalable data and machines
Implementing a similar bayesian network over a hadoop hive to make it a production level software.