The Kernel Prototype for Big Data

Rakshit Bhatt[1], Aditya Kumar Sinha[2]

[1] PG Student, GTU PG School, GTU, Ahmedabad, Gujarat, India
[2] Principal Technical Officer and Head of HPCS Group, CDAC-ACTS, Pune, Maharashtra, India

ABSTRACT

The paper presents a new class Kernel Prototype which is demonstrated to be specific for the Big Data. Although we have available systems to process the Big Data, a system purely made for the Big Data will prove effective. As an example, we consider XML file processing over the new Kernel Prototype. For this, we choose the classification of the XML files according to the tags. The simplest classification method is taken into account, which is defined by the Kernel Methods. The main reason for a new Kernel Prototype is that the currently available systems have not been made to deal with Big Data as the traditional systems are modified as per the requirement. The performance of process is the key. Thus the processing is done in the Kernel Space and in the User Space to compare the results and check the Hypothesis. Lastly the paper shows how the Kernel Method applied can help processing small files into small systems which is beneficial and initial step towards constructing the prototype for Big Data.

Keyword: *- Prototype, Kernel, Kernel Prototype, Kernel Methods, XML, Big Data*

1. INTRODUCTION.

Research in Big Data increases day by day as the data is travelling in huge amount in no time over the globe. The computational requirement for such mountainous data is huge. The current systems are modified according to the requirements [2]. But the modification is specific and limited to certain applications only. The most commonly used kernels are generally analytically derived. The problems arise when we deal with real time difficult inputs like Linear, Periodic, Radial Basis Function (RBF), Polynomial, Spline and ANOVA, with RBF being far and away the most common, and almost exclusively used for high-dimensional data [3]. Specialized techniques are used baking application kernels for either of such problems but if we look on top of that. These kernels are not so effective if used as general purpose single system kernels. An example of a general purpose method for constructing kernels is Fisher kernels [1]. Fisher kernels provide a method to construct a sensible kernel from a generative model of the data. For cases where there is a natural generative model of the data, this is a compelling and intuitive kernel. Clustering the kernels is a big task but clustering methods were designed to find “the best” partition of a dataset. To construct such group of kernels, we require group of systems connected via fast communication devices.

An important class of computations operate on voluminous data sets in ways similar to what is sometimes referred to as “Big Data” computations and other times referred to as streaming computations. These computations perform simple, straightforward, and independent operations on a large number of input data records, one chunk at a time, and can be parallelized trivially[4].

A possibility of getting grip over the large data sets is to cluster the kernels. Example, k-means is generalized to incorporate kernel functions. The shortage falls into storage of very large data sets when data is fired from the k-nodes [6]. Mathematically it can be proved that the system setup for large data sets is good enough but practically the result is always smaller than theoretical result. Yet proves better than any other primitive taken into account for computation [4].

While processing the many small files, the time increases as the size of file decreases. The computation time takes by small files is higher than the large files. This depends on the type of the file. Example, XML files being processed takes a lot of time. Hadoop is the best tool so far but the disadvantage is processing many small files takes a considerably huge amount of time. The reason is fitting the small files into the chunks and applying mapper to each file. If the number of files increases and decreases the size, the processing may end up to the hours.

For such reason, we process the XML files into the common system. We use the simplest Classification algorithm of the Kernel Methods to process the XML files in the common system to test the computation time. We thus get a positive result which encourages in constructing the prototype to further level as per the requirement and also proves the hypothesis true.

The reason constructing a prototype is that the results can be positive or negative. Also at the initial, the system is not perfect. A prototype is a raw model which can be changed anytime and anyhow if required. Also it can be used as a base to construct a real kernel if the prototype fits the requirements.

2. BACKGROUND.

A kernel is a positive semi-definite (PSD) function k (a, b): I × I → R that is used to indicate the similarity of two points in a space I [1]. Many machine learning problems can be formulated in terms of Gram matrix, rather than the more traditional design matrix. These methods are collectively referred to as kernel machines. The choice of K is critical to the predictive performance of any of these algorithms, in the same way that the choice of a sensible feature space is to traditional methods. This is because the choice of the kernel implicitly sets the feature space being used: the kernel can always be viewed as k (x.a, x.b) = φ (x.a), φ (x.b), where φ is the implicit feature space mapping.

Advantage of a kernel function is that it is sometimes possible to compute the inner product in the feature space without actually computing the potentially expensive mapping of the input points to the feature space [5]. A partition distribution naturally leads to a kernel in the following way:

Given a partition distribution P, we define the kernel

$$k_P(a,b) = \mathbb{E}\,[I[\varrho(a) = \varrho(b)]]_{\varrho \sim P}$$

to be the random partition kernel induced by P, where I is the indicator function. That is, the kernel is defined to be the fraction of the time that two points are assigned to the same cluster. Such distribution can be viewed in two ways: firstly, as a transformation that maps from a random variable to a new output space, and alternatively as a program that generates samples from the desired distribution. Thus any program that takes a dataset as input and outputs a random partition defines a partition distribution. This is the representation that will allow us to easily construct Random Partition Kernels.

3. IMPLEMENTATION.

In order to start the process, the scenario is to be setup.

- First thing is to select the algorithm for the classification of the XML files based on the tags. The Kernel Methods provide the classification algorithms. The purpose is to read the XML file content and classify it according to the fields. Thus the categorization is chosen.

  It is best suitable as the file is to be categorized according to the tag. It also acts like filtering. For example, classification of the files with the author tag. The classification can be done over various levels as per the tags. The first level processing is taken into account to reduce the complexity.

- Second step is to create the files. Various XML files can be created. The correct format can be differentiated from the incorrect format and such files can be included. In this case the file size is 1.2kb each.

As the classification is to be done only to first degree or first level, the file content is however restricted to be different. Most of the files have the author tag included in it. Also the number of files is fixed. Dynamic data is not considered as it increases the complexity to a notable figure.

- Third step is to choose a system for the experiment. Every kernel has a base processor and configuration. In this case, Intel Xeon E3 processor is used. The memory size is 16 GB. The processor has 8M Cache and works 3.70 GHz. The other features are not of the concern because the processor configuration does not play vital role. The basic information required for the experiment is stated above.

- The final step is the kernel customization and writing. This is the most important step. The kernel is partly written from scratch and partly revised from the Linux Ubuntu Kernel 3.9.3 version and the microkernel Helen Operating System.

  The Linux Kernel is the very successful kernel and it is open source project. However it still restricts to completely change the format of the project and the kernel construction must be modular. It fits into the class of monolithic kernel. It also possess mixed quality of monolithic kernel and microkernel so it can be called a hybrid kernel but an explicit patch is to applied before using it as microkernel. So it is called monolithic kernel in most of the cases. The very Linux Kernel is customized where the unwanted tasks are to be removed. The protected mode is activated which will allow direct kernel execution in the proces s.

  On the other hand, the Helen Operating system is also open source project but it falls under the class of micro kernels. This operating system has a special quality being developed for the distributed networks and performance. It provides very basis of the new kernel prototype as a reference kernel. Its idea is only taken into account. The kernel code is not used or revised before considering the process.

Thus the scenario is created before writing the code for the results.

The following code shows the XML reading files code used to read the data from the file.

```
int init_module(void)
{
   struct file *f;
   char buf[128];
   mm_segment_t fs;
   int i;
   for(i=0;i<128;i++)
      buf[i] = 0;
   printk(KERN_INFO "My module is loaded\n");
      f = filp_open("/etc/file1", O_RDONLY, 0);
   if(f == NULL)
      printk(KERN_ALERT "filp_open error!!.\n");
   else{
      fs = get_fs();
      set_fs(get_ds());
      f->f_op->read(f, buf, 128, &f->f_pos);
      set_fs(fs);
      printk(KERN_INFO "buf:%s\n",buf);
   }
   filp_close(f,NULL);
   return 0;
}
```

This opens the file from the directory. In order to check whether the file is XML or not, it has to be checked by extension.

```
bool has_txt_extension(char const *name)
```

```
{
   size_t len = strlen(name);
   return len > 4 && strcmp(name + len - 4, ".xml") == 0;
}
```

This reads the xml files only. Now the following code shows to read content of XML files and separate the tags according to names. Here the classification algorithm is referred.

```
streamFile(const char *filename)
{
   xmlTextReaderPtr  reader;
   int ret;

   reader = xmlReaderForFile(filename,  NULL,  0);
   if (reader != NULL)
   {
       ret = xmlTextReaderRead(reader);
       while (ret == 1)
      {
          processNode(reader);
          ret = xmlTextReaderRead(reader);
       }
       xmlFreeTextReader(reader);
       if (ret != 0)
      {
          fprintf(stderr,  "%s : failed to parse\n", filename);
       }
   }
   else
   {
       fprintf(stderr, "Unable to open %s\n", filename);
   }
}
```

Thus the XML content is read.

The time is noted. This process is done in 2 ways.

- Firstly check in kernel space
- Second to check in normal way

4. RESULTS.

From the above experiment we get notable results. The below table lists the performance comparison done in the kernel space and the user space.

Table 1: Performance Comparison between Kernel Space and User Space.

Processing Speed in	XML File (ms)
Kernel Space	0.17894
User Space	0.27345

The above results shows the hypothesis that the kernel space is always faster proves right.

The result is obtained from the simplest classification technique used to classify the XML tag field author from different file and then counts the number of files with author tag. This result is static as the files were written before processing. The dynamic files have not be used here as the memory storage is totally avoided.

The main reason in doing everything above is to encourage the new prototype which could be built to a real system kernel on its own. It is a first step towards experimenting the Big Data into the kernel prototype.

5. CONCLUSION

The Kernel Methods of Machine Learning provides basis for Big Data dealing with its challenges. Various techniques were referred and out of the them the best suitable classification technique in the above scenario is revised.

The XML is widely used and one of the oldest format to retrieve data and process the data from XML is a common practice. Also the XML files can be easily processed by any type of system.

The hypothesis proves correct that to gain best performance out of any system, it must be done within the system level or in the kernel space. According to the results we get a vast difference in the time taken by kernel space and user space.

The dummy kernel is just the initial step towards the construction of a new kernel specially designed for Big Data.

5. FUTURE WORK.

As of the future work, there is a lot of scope over the system.

- Implement the Machine Learning techniques in the kernel space itself and design the kernel prototype according to the Machine Learning techniques. This will increase the performance and can be scaled for the object oriented work.

- Design the kernel prototype for various Big Data fields. If the system itself is designed for the task, the task can be performed in a better way rather than customizing the system and make unnecessary changes.

- In this particular case, the number of files is limited. Dynamically the task can be performed where every new file can be processed. This way real time Big Data file processing issue can be solved. Not only for XML files, but for any log files, this method could be proved perfect.

- It is also possible to construct the prototype into a master slave approach. If any type of processing is required where one system can be made a master and govern other multiple systems and distribute the work load if necessary. This provides parallelism in the approach.

6. REFERENCES.

1. Alex Davies, Zoubin Ghahramani; “The Random forest kernel and creating other kernels for big data from random partitions; arXiv: 1402.4293v1; 18th February 2014; 1-8

2. Reza Mokhtari, Michael Stumm; “BigKernel – High Performance CPU-GPU Communication Pipelining for Big Data – style Applications”; Toronto, Canada; 2014 IEEE 28th International Parallel & Distributed Processing Symposium; 819-828.

3. Brian Kulis; “Scalable Kernel Methods for Machine Learning” University of Texas, Austin, 2008; 6-11

4. Raghvendra Mall, Rocco Langone, Johan Suykens; "Kernel Spectral Clustering for Big Data Networks"; Entropy, 2013; 1567-1586.

5. Ali Rahimi, Benjamin Recht; "Random Features for Large - Scale Kernel Machines"; IEEE International Conference on Machine Learning, 2008; 1-10