GSoC 2015 Application: Ayush Pandey: Fast Interactive Active Machine Learning Enabled by GPUs

About Me

Username and Contact Information

Name : Ayush Pandey
University : Indian Institute of Technology, Kharagpur
email : ayushpandey.iitkgp@gmail.com
IRC Handle : krishnakanhaiya at freenode.net
github username : Ayush-iitkgp

Personal Background

I am a junior undergraduate student at IIT Kharagpur, India. I'm pursuing an Integrated Master of Science degree in Mathematics and Computing offered by the Department of Mathematics. I'm proficient in C, C++, Java and Python.

Previous Work Experience

Internship

In the summers of 2014, I interned at Flutura Decision Sciences and Analytics where I worked with the R&D team of the company on the project entitled "Large Scale in-memory real-time analytics on Apache Spark Framework"[Technical Report]. As a Data Scientist intern, I was responsible for implementing and benchmarking the accuracy and the running time of parallel/distributed programming paradigms like MapReduce on Hadoop and Resilient Distributed Datasets in Spark.

Data Analytics

In my junior year, I took part in two competitions related to Data Analytics :-

- Theft Analytics (Anomaly Detection) by an Electricity Company (our team won gold medal).
- Predicting the likelihood of the person sinking during titanic disaster (OpenIIT Data Analytics).

Online Courses

- Machine Learning
- Introduction to Data Science
- Heterogeneous Parallel Programming(ongoing-week 6)[Codes]

The Project

The Problem and Motivation

The aim of the project is to implement active learning Support Vector Machine classification algorithm on GPU and providing a web framework for collecting user/expert feedback whenever a new ambiguous sample arrives. Active Learning strategy to be used for training SVM is Pool-Based Learning for Uncertainty Sampling. I am proposing two approaches for SVM training on the GPU

1. Using LIBSVM for training and using GPU to accelerate the parts of the computation, most notably the computation of the RBF Kernel[7]. This approach gives more accurate result as compared to the other method but takes slightly more time.
2. Not using the training method provided by LIBSVM(which runs on CPU) instead implementing the Sequential Minimal Optimization Algorithm[3] with first-order variable selection strategy proposed by Keerthi on GPU. But this method has the accuracy issues because most of the GPUs support only single precision float arithmetics.

Why use Machine Learning for Cancer Research ?

The conventional approach in cancer classification is based on the morphological appearance of the tumour which has the following disadvantages:-
1- High bias by experts in identifying the type of tumour.
2- Difficulty in identifying the different types of cancer because most of the cancers have almost same biological behaviour.

Hence, a new method of classification at the genotype level is performed where we apply the machine learning models to learn from the gene expression profiles of various cancerous samples and classify the new sample as cancerous or non-cancerous.

Why Active Machine Learning ?

Getting the genotype profile of a cell involves very low or no cost but identifying them as cancerous or non-cancerous involves a series of medical test which incurs a very high cost (in terms of time and money). Thus if we want to use Passive Machine Learning then first we will have to obtain considerable amount of cancerous and non-cancerous samples which in turn result in high training cost (in terms of time as well as money). Instead we start with just one cancerous and one non-cancerous example to train our model initially and if a new sample arrives we check if it is similar to the already present labelled samples, if yes we do not perform test on it and if no then we handover the sample to the expert for labelling it and again train our model with new data. This in turn decreases the training cost while the performance of our model remains almost same or better than the passive learning model.

Why Active Learning using GPU not CPU ?

Due to the limitations of the processing power of the semiconductor the CPU processing speed has more or less become constant with time. But the data size is increasing so we need a better computation system which can grow itself according to the data size, this is provided by GPU which are basically collection of threads operating independently and parallely and can grow in number with the increase in data size.

Why only Support Vector Machine ?

While going through the literature[1], the only algorithm that is found suitable for cancer classification using gene expression data is Support Vector Machine.

The Plan

I propose to implement the Active Support Vector Machine Learner on GPUs and develop web framework so that the expert interactively label the new/ambiguous samples.

The Algorithms and the Mathematics Used

Support Vector Machine Definition:-

Mathematical Formulation of the Support Vector Machine

Let N=No. of training samples

Primal Form

$$\mathbf{min} \; ||\mathbf{w}||^2 + \mathbf{C} \sum_{i=1}^{i=N} \boldsymbol{\xi}_\mathbf{i}$$

$$\mathbf{subject\ to}$$

$$y_i \, (w \cdot x_i - b) \geq 1 - \xi\mathbf{i}, \quad \forall i \in \{1,2,3....N\}$$

where C is the regularization parameter used to avoid overfitting as well as the underfitting and $\xi\mathbf{i}$ penalizes the failure of an example to reach the correct margin.

Dual Form

$$\mathbf{min}\ \tfrac{1}{2} \sum_{i=1}^{i=N} \sum_{j=1}^{j=N} y_i y_j K(x_i,x_j) \; \alpha_i \alpha_j - \sum_{i=1}^{i=N} \alpha_i$$

$$\mathbf{subject\ to}$$

$$0 \leq \alpha_i \leq C, \; \forall i \in \{1,2,3....N\}$$

$$\sum_{i=1}^{i=N} y_i \alpha_i = 0$$

where w is the weight vector, K is the Kernel Function(Linear, Polynomial, Gaussian) and α_i are the lagrangian multiplier corresponding to each training instance.

Note: The dual form is the Quadratic Optimization problem and solving this using conventional method is computationally very expensive therefore we use Sequential Minimization Optimization Algorithm described below to solve the above quadratic problem in very less time and memory space.

Active Learning with Support Vector Machine

There are mainly two approaches to active Learning[2] namely:

1. **Uncertainty Sampling:-** In this approach, we query (i.e ask an expert to label it) instances the learner/model is most uncertain about.
2. **Hypothesis Space Search:-** In this approach, we query the instance for which majority of the learners/models disagree with each other.

I plan to implement Uncertainty Sampling Approach to Active machine Learning.

Now there are two ways in which Uncertainty Sampling is used in practice:

1. Selective Sampling
2. Pool Based Active Learning

I will be using Pool Based Active Learning in my project.

Pool Based Active Learning Approach

ALGORITHM

1. Build an initial classifier **with at least one positive and one negative example**.
2. While an expert can label example
    a. Apply the current classifier to each unlabeled data
    b. Find m examples which are most ambiguous for the classifier
    c. Ask the expert to label the m examples
    d. Train a new model on all labelled examples

Q:- How to select m examples from the pool of unlabelled data ?
Ans:- Warmuth[6] tested different selection strategies and showed that strategy in which we select the examples with the largest predicted values is the best strategy to find many positive examples in very few iteration.

Q:- What should be the size of m to be chosen in each iteration?
Ans:- A large value of m is used to achieve good performance of the classifier in very few iteration[1].

APPROACH 1

In this method, we decompose the SVM training problem in two different tasks:-

1. The calculation of the Kernel Matrix to be executed on the GPU

2. The core SVM task of decomposing and solving the series of the quadratic problems to be executed on CPU using LIBSVM modules.

Algorithm to Compute the Kernel Matrix

ALGORITHM

1. Precalculate on the CPU, the sum of the squares of the elements for each training vector (i.e $x_i.x_i$)
2. Convert the training vectors array into column wise format
3. Allocate memory on the GPU for the training vectors array.
4. Load the training vectors array, in the translated format to the GPU memory.
5. FOR (each training vector) DO
    a. Load the training vector to the GPU (because the version on the GPU is in a translated format).
    b. Perform the matrix-vector multiplication, i.e. calculate the dot products, using CUBLAS.
    c. Retrieve the dot products vector from the GPU.
    d. Calculate the row of the Kernel Matrix by adding the training vector squares, then calculating $K(x_i, x_j)$.

   END DO
6. De-allocate memory from GPU.

Modules to be used and their Description

The Host Module:

svm.cpp- This module will contain the core SVM implementation used by LIBSVM. It will contain the functions for SMO Solver Algorithm which is already implemented in LIBSVM.(The code is provided by LIBSVM . Copyright (c) 2000-2014 Chih-Chung Chang and Chih-Jen Lin).

```
class Solver {
public:
	Solver() {};
	virtual ~Solver() {};

	struct SolutionInfo {
		double obj;
		double rho;
		double upper_bound_p;
		double upper_bound_n;
	};

	void Solve(int l, const QMatrix& Q, const double *p_, const schar *y_,
		   double *alpha_, double Cp, double Cn, double eps,
		   SolutionInfo* si, int shrinking);
```

```
protected:
	int active_size;
	schar *y;
	double *G;		// gradient of objective function
	enum { LOWER_BOUND, UPPER_BOUND, FREE };
	char *alpha_status;	// LOWER_BOUND, UPPER_BOUND, FREE
	double *alpha;
	const QMatrix *Q;
	const double *QD;
	double eps;
	double Cp,Cn;
	double *p;
	int *active_set;
	double *G_bar;		// gradient, if we treat free variables as 0
	int l;
	bool unshrink;

	double get_C(int i)
	{
		return (y[i] > 0)? Cp : Cn;
	}
	void update_alpha_status(int i)
	{
		if(alpha[i] >= get_C(i))
			alpha_status[i] = UPPER_BOUND;
		else if(alpha[i] <= 0)
			alpha_status[i] = LOWER_BOUND;
		else alpha_status[i] = FREE;
	}
	bool is_upper_bound(int i) { return alpha_status[i] == UPPER_BOUND; }
	bool is_lower_bound(int i) { return alpha_status[i] == LOWER_BOUND; }
	bool is_free(int i) { return alpha_status[i] == FREE; }
	void swap_index(int i, int j);
	void reconstruct_gradient();
	virtual int select_working_set(int &i, int &j);
	virtual double calculate_rho();
	virtual void do_shrinking();
private:
	bool be_shrunk(int i, double Gmax1, double Gmax2);
};
```

The Device Module

kernelmatrix.cu- This module will use the CUBLAS library provided by NVIDIA to pre-calculate the Kernel matrix to be passed to the svm.cpp module.

```
#include "/usr/local/cuda/include/cublas.h"
```

```
void kernelmatrix( struct svm_problem *prob, struct svm_problem *pecm, float
*gamma  )
// gamma is the same as used by Gaussian Kernel, struct svm_problem describes
the entire training data
{
   // Calculating the sum of the squares of the elements of for each training
vector
      for( i_r=0; i_r < ntv ;i_r++)
      {
            for ( i_c = 0; i_c < len_tv; i_c++)
                        tva[i_r * len_tv + i_c] =
(float)prob->x[i_r].values[i_c];
      }

// Loading CUBLAS
cublasInit();

//Allocating the training vector the GPU
cublasAlloc(len_tv, sizeof(float), (void**)&g_vtm);

// Calculating the dot product xixj
      for ( trvei = 0; trvei < ntv; trvei++)
      {
            cublasSetVector(len_tv, sizeof(float), &tva[trvei*len_tv], 1,
g_vtm, 1);

            cublasSgemv('n', ntv, len_tv, 1, g_tva, ntv , g_vtm, 1, 0,
g_DotProd, 1);

            cublasGetVector(ntv, sizeof(float), g_DotProd, 1, DP, 1);

// Calculating the each row of the Kernel Matrix
      for( i_c =0; i_c< ntv; i_c++ )
            {
                  v_f_g[i_c] =
                        exp( -g_val * (tv_sq[trvei] + tv_sq[i_c]-((double)2.0)*
(double)DP[i_c] ));
            }

                  pecm->x[trvei].values[0] = trvei+1;
                  for( i_c =0; i_c< ntv; i_c++ )
                  {
                        pecm->x[trvei].values[i_c+1] = v_f_g[i_c];

                  }

      }
    }
```

APPROACH 2

Sequential Minimal Optimization Algorithm with first order variable selection for training Support Vector Machine

Using this iterative approach for training SVM, we solve the SVM QP problem without using the numerical QP optimization step thus significantly improve the training time. In this approach we decompose the main QP problem into small QP sub-problem to optimize only the subproblems involving two lagrangian multipliers, the choice of the required two lagrangian multiplier is given by Keerthi Algorithm[4](which is described below). Hence, in an iteration we only optimize that QP sub-problem which involve only these two lagrangian multipliers.

Variable selection using Keerthi Algorithm

Define $f_i = \sum_{j=1}^{j=n} \alpha_j y_j K(x_i, x_j) - y_i$

ALGORITHM

1. Partition the training data points into 5 sets, represented by their indices:
   a. (Unbound SVs) $\mathbf{I_0} = \{i : 0 < \alpha_i < C\}$
   b. (Positive Non SVs) $\mathbf{I_1} = \{i : y_i > 0, \alpha_i = 0\}$
   c. (Bound Negative SVs) $\mathbf{I_2} = \{i : y_i < 0, \alpha_i = C\}$
   d. (Bound Positive SVs) $\mathbf{I_3} = \{i : y_i > 0, \alpha_i = C\}$
   e. (Negative Non SVs) $\mathbf{I_4} = \{i : y_i < 0, \alpha_i = 0\}$
2. Find two lagrangian multiplier α_{Ilow} and α_{Ihigh} whose indices are given by:
   a. $I_{high} = \mathbf{arg}_{\mathbf{min\ i \in I0 \cup I1 \cup I2}} \mathbf{f_i}$
   b. $I_{low} = \mathbf{arg}_{\mathbf{min\ i \in I0 \cup I3 \cup I4}} \mathbf{f_i}$

Thus α_{Ihigh} and α_{Ilow} are the two lagrangian multiplier to be optimized in an iteration when using SMO.

Sequential Minimization Optimization Algorithm

Define
$b_{high} = \min \{f_i : i \in I_0 \cup I_1 \cup I_2\}$
$b_{low} = \max \{f_i : i \in I_0 \cup I_3 \cup I_4\}$
$\alpha'_{Ilow} = \alpha_{Ilow} + (y_{Ilow} (b_{high} - b_{low})) / \eta$
$\alpha'_{Ihigh} = \alpha_{Ihigh} + y_{Ilow} y_{Ihigh} (\alpha_{Ilow} - \alpha_{Ilow})$

(where $\eta =$)

Note:- α'_{Ilow} and α'_{Ihigh} are the updated values of the two lagrangian multipliers α_{Ilow} and α_{Ihigh} which are chosen at the beginning of each iteration.

ALGORITHM

Input:- Training data x_i and its corresponding label y_i , $\forall i \in \{1,2,3....N\}$

1. Initialize: $\alpha_i = 0$, $f_i = -y_i$, $\forall i \in \{1,2,3....N\}$
2. Compute: b_{high}, I_{high}, b_{low}, I_{low}
3. Update α_{Ihigh} and α_{Ilow}
4. **repeat**
    a. Update f_i , $\forall i \in \{1,2,3.....N\}$
    b. Compute: b_{high}, I_{high}, b_{low}, I_{low}
    c. Update α_{Ihigh} and α_{Ilow}

   until $b_{low} \leq b_{high} + 2\epsilon$ where ϵ is the tolerance of termination criterion.

Output:- The optimized values of α_i $\forall i \in \{1,2,3....N\}$ for which the optimal solution of the given QP occurs.

Predicting the outcome of an instance

We use use the output of the SMO Algorithm to predict the label($\hat{Z}$) for a new instance given by whose feature vector is given by z

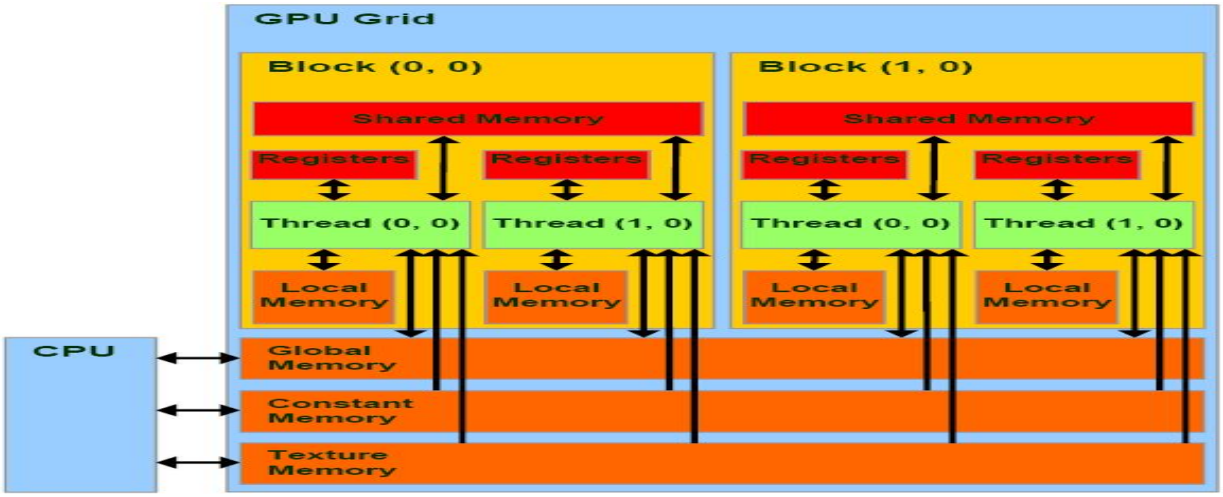
$$\hat{Z} = \text{sgn} \{ b + \sum_{i=1}^{i=N} y_i \alpha_i K(x_i , z) \}$$

Implementation Details

In this section, I am going to elaborate how I am going to implement the Above Sequential Minimal Optimization problem on using GPU as well as the CPU (i.e. using CPU for the sequential and GPU for the parallel part of the Algorithm). I am going to begin by giving a very brief introduction to the architecture of the GPUs.

GPU Architecture

The threads are the building blocks of the GPU. Each thread itself has its own Arithmetic Logical Unit, register and the local memory and hence can operate independently. The threads are organised in a block,threads within a block can interact via shared memory, atomic operations(i.e. update the same variable)and barrier synchronization(i.e. they can wait for each other). Threads in different block do not interact. All the threads of the GPU are connected to Global memory and the host memory(i.e. CPU is only connected to the GPU Global Memory) Finally, the blocks as well as the threads can be 1-D, 2-D or 3-D.



CUDA THREAD ORGANIZATION(Credits: https://www.evl.uic.edu/aej/525/lecture04.html)

Programming Considerations

Below are the key point we have to keep in mind while programming the SMO Training Algorithm on the GPU[5] to extract maximum performance:-

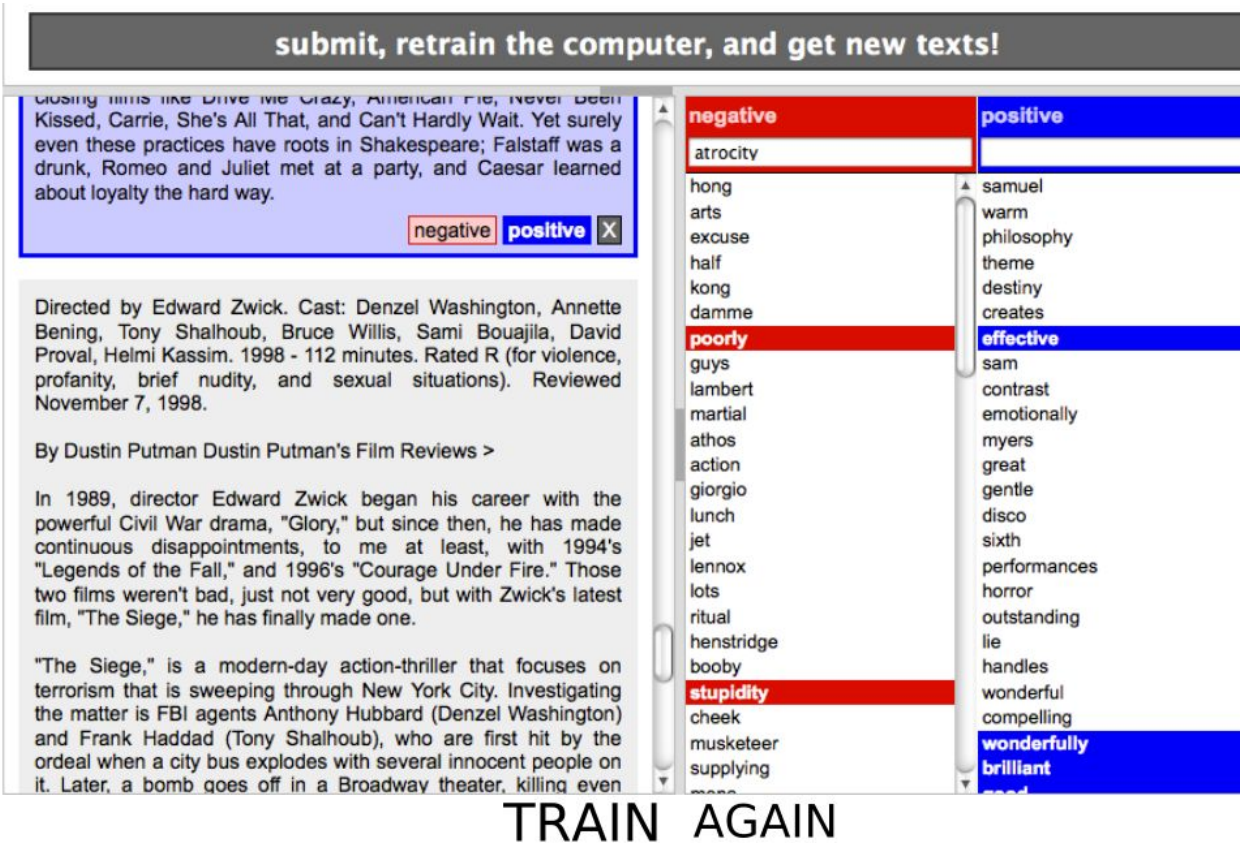
1. The parallelism needed for maximizing the performance of the GPU is derived from the number of training instances in the data set i.e. **if we have N training points we are going to create N threads in the GPU**.
2. In CUDA two blocks do not share common memory and they can't directly communicate to each other. Since there is restriction on the number of threads each block (i.e each block can have only 512 threads) therefore if the training data has more than 512 instances, we will have more than one block.
3. From the above algorithm, we know that **we have to calculate f_i'** for each point in the data set **during every iteration** and this calculation is done by the thread to which that point is allotted hence all the **f_i'** are calculated for each point **simultaneously.**
4. Every iteration of the SMO Algorithm can be thought of as an MapReduce step where in the **map phase we calculate f_i' and in the reduce phase we compute b_{high}, I_{high}, b_{low}, I_{low}.**
5. There will be exactly one map phase but the number of reduce phases will be more than one. **In first reduce phase we find b_{high}, I_{high}, b_{low}, I_{low} which is local to each block and in the second reduce phase we find the global b_{high}, I_{high}, b_{low}, I_{low}. This global b_{high}, I_{high}, b_{low}, I_{low} is what we use in the above algorithm**.
6. If we look at the above SMO Algorithm, we find that we have to compute $K(x_i,x_j)$ $\forall i,j \in \{1,2,3....N\}$ and this in turn forms a very significant part of the computation. So rather than calculating $K(x_i,x_j)$ at every iteration, **we calculate $K(x_i,x_j)$ just once and cache it in the available memory in the GPU**.
7. Since the size of GPU memory is not very large so if the size of the training data is very large such that the GPU can't store the Kernel value **$K(x_i,x_j)$ for each** $i,j \in \{1,2,3....N\}$ then in such case during every iteration CPU checks which of the values are present in the

cache, if the required $K(x_i,x_j)$ is not present in the cache we remove the recently used $K(x_i,x_j)$ from the cache and needed $K(x_i,x_j)$ is added to the GPU cache.

8. Alternate approach for 7 is to calculate self dot product on the CPU and transfer the result to the GPU. Then during training iterations use NVIDIA's CUDA basic linear algebra Library CUBLAS to compute the necessary $x_i.x_j$ and then use a helper function to subtract two vectors, multiply a vector by λ, and exponentiate.

The Web Framework

Taking inspiration from the Burr Settles CMU Lecture, I propose to develop a similar web framework. The GUI will provide the option for the expert to mark an new ambiguous sample sample as negative or positive which will be connected to the backend . So when the expert hits the "**TRAIN AGAIN"** button, new model will be generated based on the instances/samples labelled so far.



Courtesy:- Burr Settles- Dualist

Timeline

28th April-24th May :- Bonding Period as well as acquiring the expert level knowledge of programming in CUDA.
Week 1,2: I plan to use this week to read and understand how similar projects have been planned and executed. During this week, I will be creating the rough sketch of the modules to be used in our project and establish a relationship between them. The expected outcome is the Design Specification of the project.
Week 2: In this week, I will be implementing the first approach to developing a GPU enabled Machine Learning System. During the week, I will separately write and integrate the CPU and the GPU modules. The expected outcome of this week is the full fledged Active Machine Learning system.
Week 3: This will be utilized for testing and benchmarking the running time and the accuracy of the algorithm. I also plan to use this week to write a pseudo-code for second method.
Week 4: This week will be fully devoted to implementing the second method. In this week, I plan to implement the SMO Solver on the GPU. Also I will try to implement the cross-validation module on the GPU.
Week 5: The GPU modules developed in week 4 will be integrated with the CPU modules. Also, the running time and the accuracy of the two methods will be compared and benchmarked.
Week 6: In this week, rather than writing new modules, I will try to optimize my code to make it faster and platform independent.
Week 7: This week will be utilized for developing the web framework of the project. Mainly the focus will be on the front end. Also, a lot of reading will be done in this week on how to integrate the back end with the front end.
Week 8: This week will be devoted to integrating and testing the web framework. A few examples will be tried and based on the requirement and the advice of the mentor changes in the GUI will be made.

References

[1]. Active Learning with Support Vector Machine Applied to Gene Expression Data for Cancer Classification
[2]. Active Learning Literature Survey
[3]. Fast Training of Support Vector Machines using Sequential Minimal Optimization
[4]. Improvements to Platt's SMO Algorithm for SVM Classifier Design
[5]. Fast Support Vector Machine Training and Classification on Graphics Processors
[6]. Active learning with support vector machines in the drug discovery process
[7]. GPU ACCELERATION FOR SUPPORT VECTOR MACHINES