Parallel Fuzzy Rough Support Vector Machine for Data Classification in Cloud Environment

Arindam Chaudhuri
Samsung R & D Institute Delhi Noida - 201304 India
E-mail: arindamphdthesis@gmail.com

Keywords: FRSVM, Hadoop, MapReduce, PFRSVM

Received: November 27, 2014

Data classification has been actively used for most effective means of conveying knowledge and information to users. With emergence of huge datasets existing classification techniques fail to produce desirable results where the challenge lies in analyzing characteristics of massive datasets by retrieving useful geometric and statistical patterns. We propose a supervised parallel fuzzy rough support vector machine (PFRSVM) for in-data classification in cloud environment. The fuzzy rough set model takes care of sensitiveness of noisy samples and handles impreciseness in training samples bringing robustness to results. The algorithm is parallelized with a view to reduce training times. The system is built on support vector machine library using Hadoop implementation of MapReduce. The algorithm is tested on large datasets present at the cloud environment available at University of Technology and Management, India to check its feasibility and convergence. It effectively resolves outliers' effects, imbalance and overlapping class problems, normalizes to unseen data and relaxes dependency between features and labels with better average classification accuracy. The experimental results on both synthetic and real datasets clearly demonstrate the superiority of the proposed technique. PFRSVM is scalable and reliable in nature and is characterized by order independence, computational transaction, failure recovery, atomic transactions, fault tolerant and high availability attributes as exhibited through various experiments.

Povzetek: Razvita je nova verzija metode podpornih vektorjev (tj. strojnega učenja) nad velikimi podatki v oblaku, imenovana PFRSVM.

1 Introduction

The volume of business data is always expanding with rapid increase of global competitiveness [1] among the organizations. It is estimated that the volume of business data double within every two years. This fact is evident in both advanced and emerging economies. A common task often performed by the analysts and managers is data classification [2] which categorizes data into different subgroups in which ideas and objects are recognized and understood. In this process relevant and meaningful hidden information is discovered from data. From economic perspective, knowledge obtained from classified data can be applied directly for business application and services. However, as the amount of data increases continuously classification becomes more and more complex [3] where present techniques produce spurious results. This in turn disturbs the integrity of data. The inherent challenge lies in analyzing and interpreting characteristics of huge datasets by extracting significant usage patterns through various machine learning techniques [3].

Knowledge discovery [4] of meaningful information has been a topic of active research since past few years. The ongoing rapid growth of online data generally referred to as big data [1] have created an immense need for effective classification techniques [5]. The process of extracting knowledge from data draws upon the research in pattern classification and optimization to deliver advanced business intelligence [3]. The big data is specified using three characteristics viz. volume, variety and velocity [6]. This means that at some point in time when volume, variety and velocity of data are increased the current techniques may not be able to process the data. Ideally these three characteristics of a dataset increase data complexity and thus the existing techniques function below expectations within given processing time. Many applications such as classification, risk analysis, business forecasting etc. suffer from this problem. These are time sensitive applications and require efficient techniques to tackle the problem on the fly.

Some emerging techniques such as hadoop distributed file systems [7], cloud technology [8] and hive database [9] can be combined to big data classification problem. With this motivation this work entails the development of a supervised classification algorithm in cloud environment incorporating machine intelligence techniques for mining useful information [10]. The classification here is performed by the fuzzy rough [11] version of support vector machine (SVM) [12] which though considered faster than artificial neural network (ANN) [13] for training large datasets but is computationally intensive. Given a large enough dataset the training time can range from days to weeks. This problem is handled by extending the fuzzy

rough version of SVM to parallel framework using hadoop implementation of MapReduce.

In this Paper, we propose parallel fuzzy rough support vector machine (PFRSVM) with MapReduce to classify huge data patterns in a cloud environment. Using MapReduce the scalability and parallelism of split dataset training is improved. Fuzzy rough support vector machine (FRSVM) [12], [14] is trained in cloud storage servers that work concurrently and then in every trained cloud node all support vectors are merged. This operation is continued until the classifier converges to an optimal function value in finite iteration size. This is done because it is impossible to train large scale datasets using FRSVM on a single computer. The fuzzy rough model is sensitive to noisy mislabeled samples which brings robustness to classification results. All the experiments are performed on the cloud environment available at University of Technology and Management, India.

The major contributions of this work include: (a) parallel implementation of FRSVM in cloud environment (b) formulation of sensitive fuzzy rough sets for noisy mislabeled samples to bring robustness in classification results (c) training FRSVM with MapReduce (d) identifying relevant support vectors at each computing node and merging with global support vectors (e) development of a scalable and reliable in-stream data classification engine adhering to the fundamental rules of stream processing such that it is maintains order independence in data processing, streamlines computational transaction, recovers from failure, generates atomic transactions and are fault tolerant and highly available in nature. To the best of our knowledge PFRSVM presented in this research work illustrates a robust architecture of in-stream data analytics which is first of its kind. The proposed computational framework has never been studied rigorously prior to this research work [15].

This Paper is organized as follows. The section 2 presents some work related to classification using fuzzy and rough versions of SVM. In section 3 an overview of SVM is presented. This is followed by a brief discussion on fuzzy rough sets. FRSVM is described in section 5. The section 6 illustrates the MapReduce pattern. PFRSVM formulation is highlighted in section 7. The experimental results and discussions are given in section 8. Finally conclusions are given in section 9.

2 Related work

Over the past decade data classification though fuzzy and rough versions of SVM have been rigorously used by researchers in several applications [15]. A brief illustration of few important ones is highlighted here. Mao et al investigated multiclass cancer classification by using fuzzy support vector machine (FSVM) and binary decision tree with gene selection. They proposed two new classifiers with gene selection viz. FSVM and binary classification tree based on SVM tested on three datasets such as breast cancer, round blue cell tumors and acute leukemia data which gave higher prediction accuracy. Abe et al studied multiclass problems using FSVM where they used truncated polyhedral pyramidal membership function for decision functions to train SVM for two different pairs of classes. Huang et al proposed new SVM fuzzy system with high comprehensibility where SVM is used to select significant fuzzy rules directly related to a fuzzy basis function. Analysis and comparative tests about SVM fuzzy system show that it possesses high comprehensibility and satisfactory generalization capability. Thiel et al studied fuzzy input fuzzy output one against all SVM where fuzzy memberships were encoded in fuzzy labels to give fuzzy classification answer to recognise emotions in human speech. Shilton et al proposed an iterative FSVM classification whereby fuzzy membership values are generated iteratively based on positions of training vectors relative to SVM decision surface itself. Li et al studied fault diagnosis problem using FSVM. Pitiranggon et al constructed a fuzzy rule based system from SVM which has the capability of performing superior classification than the traditional SVM. Zhu et al used FSVM control strategy based on sliding mode control to reduce oscillation. Parameters of FSVM controller were optimized by hybrid learning algorithm which combines least square algorithm with improved genetic algorithm to get the optimal control performance for controlled object. Li et al proposed double or rough margin based FSVM algorithm by introducing rough sets into FSVM. First, the degree of fuzzy membership of each training sample is computed and then data with fuzzy memberships were trained to obtain decision hyperplane that maximizing rough margin method. Chen et al extracted a new feature of consonants employing wavelet transformation and difference of similar consonants. Then algorithm classified consonants using multiclass FSVM. Long et al proposed network intrusion detection model based on FSVM. They concentrated on automatic detection of network intrusion behavior using FSVM. The system composed of five modules viz. data source, AAA protocol, FSVM located in local computer, guest computer and terminals. The intrusion detection algorithm based on FSVM is implemented by training and testing process. Jian et al coined FSVM based method to refine searching results of SEQUEST which is a shotgun tandem mass spectrometry based peptide sequencing using programs on a dataset derived from synthetic protein mixtures. Performance comparison on various criteria show that proposed FSVM is a good approach for peptide identification task. Duan et al studied FSVM based on determination of membership. They investigated sensitivity issues relating SVM to outlier and noise points which favours use of FSVM though appropriate fuzzy membership. Shi et al proposed an emotional cellular based multiclass FSVM on product's kansei image extraction. Li et al proposed fuzzy twin SVM algorithm that has computational speed faster than traditional SVM. It takes into account the importance of training samples on learning of decision hyperplane with respect to classi-

fication task. Yan et al proposed probability FSVM based on the consideration both for fuzzy clustering and probability distributions. The model is based on consideration that probability distribution among samples exhibits superior performance.

3 Support vector machine

Support vector machine (SVM) [12] is a promising pattern classification tool based on structural risk minimization and statistical learning theory [16]. Many complex problems have been solved by SVMs. It minimizes prediction error and models complexities. SVM formalizes classification boundary by dividing points having different labels so that boundary distance from closest point is maximized. It transforms training vectors into high dimensional feature space labeling each vector by its class. It classifies data through set of support vectors which are members of training input set that outline a hyperplane in feature space [12], [16] as shown in figure 1. Structural risk minimization reduces generalization error. The number of free parameters depends on margin that separates data points. SVM fits hyperplane surface to data using kernel function that allows handling of curse of dimensionality. To recognize support vectors the problem is restructured as following quadratic optimization problem [12] which is convex, guarantees uniqueness and optimality:

$$\min \quad \|w\|^2$$

$$\text{subject to: } z_i\left(w^T Y_i + b\right) \geq 1,\ i = 1, \ldots\ldots, M \quad (1)$$

In equation (1) is weight vector and b is bias term. Slack variables $\xi_i; i \in \{1, ..., M\}$ measures violation of constraints such that the quadratic problem now becomes:

$$\min \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{M}\xi_i$$

$$\text{subject to:}$$
$$\left\{z_i\left(w^T Y_i + b\right)1 - \xi_i; i = 1, \ldots\ldots, M\ ;\ \xi_i \geq 0 \right. \quad (2)$$

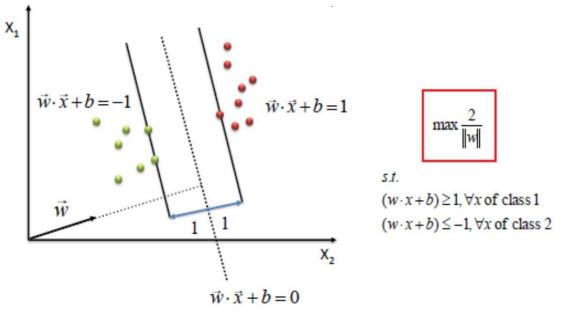

Figure 1: Separating Hyperplane between Classes leading to different Support Vectors.

In equation (2) regularization parameter C determines constraint violation cost. To classify nonlinear data [17] the mapping transforms classification problem into higher dimensional feature space giving linear separability. This is achieved by transforming Y_i into higher dimensional feature space through $\Phi(Y_i)$ satisfying Mercer's condition [12]. The quadratic problem is solved by scalar product $K(Y_i, Y_j) = (Y_i) \cdot (Y_j)$. By using Lagrange multipliers and kernels the problem becomes:

$$\min \quad \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M} z_i z_j \alpha_i \alpha_j K(Y_i, Y_j) - \sum_{j=1}^{M}\alpha_j$$

$$\text{subject to:}$$
$$\sum_{i=1}^{M} z_i\alpha_i = 0; i = 1, \ldots\ldots, M;\ 0 \leq \alpha_i \leq C \quad (3)$$

The commonly used kernels are polynomial and gaussian functions. In training SVMs we need kernel function and its parameters to achieve good results and convergence. When solving two class classification problems each training point is treated equally and assigned to only one class. In many real word problems some training points are corrupted by noise. Some points also are misplaced on wrong side. These points are outliers and belong to two classes with different memberships. SVM training algorithm makes decision boundary to severely deviate from optimal hyperplane as it is sensitive to outliers [12], [16]. This is handled by techniques as illustrated in [17], [18]. [19].

4 Fuzzy rough sets

Let R be an equivalence relation on universal set P. The family of all equivalence classes induced on P by R is denoted by $\frac{P}{R}$. One such equivalence class in $\frac{P}{R}$ contains $p \in P$ is denoted by $[p]_R$. For any output class $A \subseteq P$ lower and upper approximations approaching A closely from inside and outside are defined [11]. Rough set $R(A)$ is a representation of A by lower and upper approximations. When all patterns from equivalence class do not carry same output class label rough ambiguity is generated as manifestation of one-to-many relationship between equivalence class and output class labels. The rough membership function $rm_A(p) : A \rightarrow [0, 1]$ of pattern $p \in P$ for output class A is given by equation (4) in Appendix A.

When equivalence classes are not crisp they form fuzzy classes $\{FC_1, \ldots.., FC_H\}$ generated by fuzzy weak partition [11] of input set P. Fuzzy weak partition means that each $FC_i; i \in \{1, \ldots\ldots, H\}$ is normal fuzzy set. Here, $\max_p \mu_{FC_i}(p) = 1$ and $inf_p \max_i \mu_{FC_i}(p) > 0$ while $\underbrace{sup}_{x} \min_{i,j}\left\{\mu_{FC_i}(p), \mu_{FC_j}(p)\right\} < 1 \quad \forall i, j \in \{1, 2, \ldots\ldots, H\}$. Here $\mu_{FC_i}(p)$ is fuzzy membership function of pattern p in class FC_i.The output classes $C_c; c = \{1, 2, \ldots\ldots, H\}$ may be fuzzy also. Given a weak fuzzy partition $\{FC_1, FC_2, \ldots\ldots, FC_H\}$ on

P description of any fuzzy set C_c by fuzzy partitions under upper approximation $\overline{C_c}$ is given by equation (5) in Appendix A and lower approximation $\underline{C_c}$ is:

$$\mu_{\underline{C_c}}(FC_i) = \underbrace{sup}_{x \in C_c} min\left\{\mu_{FC_i}(p), \mu_{C_c}(p)\right\} \quad \forall p \quad (6)$$

The tuple $\left\langle \underline{C_c}, \overline{C_c} \right\rangle$ is called fuzzy rough set. Here $\mu_{C_c}(p) = \{0,1\}$ is fuzzy membership of input p to C_c. The fuzzy roughness appears when class contains patterns that belong to different classes.

5 Fuzzy rough support vector machine

Based on SVM and fuzzy rough sets [11] we present FRSVM. To solve misclassification problem in SVM, fuzzy rough membership is introduced to each input point such that different points can make unique contribution to decision surface. The input's membership is reduced so that its contribution to total error term is decreased. FRSVM also treats each input as of opposite class with higher membership. This way fuzzy rough machine makes full use of data and achieves better generalization ability. We consider training sample points as:

$$SP = \{(Y_i, z_i, frm_i(p)); i = 1, \ldots\ldots, M\} \quad (7)$$

Here each $Y_i \in \mathrm{R}^N$ is training sample and $z_i \in \{-1, +1\}$ represents its class label; $frm_i(p); i = 1, \ldots\ldots, M$ is fuzzy rough membership function satisfying $s_j \leq frm_i(p) \leq s_i; i, j = 1, \ldots\ldots, M$ with sufficiently small constant $s_j > 0$ and $s_i \leq 1$ considering pattern p. Taking $P = \{Y_i \mid (Y_i, z_i, frm_i(p)) \in SP\}$ containing two classes; one class C^+ with sample point $Y_i(z_i = 1)$ and other class C^- with sample point $Y_i(z_i = -1)$ such that:

$$C^+ = \{Y_i | Y_i \in SP \wedge z_i = 1\} \quad (8)$$

$$C^- = \{Y_i | Y_i \in SP \wedge z_i = -1\} \quad (9)$$

Here, $P = C^+ \cup C^-$; then classification problem is given by equation (10) in Appendix A.

In equation (10) C is constant. The fuzzy rough membership $frm_i(p)$ governs the behavior of corresponding point Y_i towards one class and ξ_i is error measure in FRSVM. The term $frm_i(p)\xi_i$ is an error measure with different weights. A smaller $frm_i(p)$ reduces the effect of ξ_i in equation (10) such that point Y_i is treated less significant. The quadratic problem can also be solved by their dual alternatives [12]. The kernel function used is hyperbolic tangent kernel $K(Y_i, Y_j) = \tanh\left[(Y_i) \cdot (Y_j)\right]$ [13] given in figure 2. It is conditionally positive definite and allows lower computational cost and higher rate of positive eigenvalues of kernel matrix alleviating limitations of other kernels. The sigmoid kernel has been used in several cases with appreciable success [19] motivating its usage in fuzzy rough membership function in proposed machine. The class centre of C^+ and C^- in feature space is defined as Φ_+ and Φ_- respectively.

$$\Phi_+ = \frac{1}{m_+} \sum_{Y_i \in C^+} (Y_i) f_i \quad (11)$$

$$\Phi_- = \frac{1}{m_-} \sum_{Y_i \in C^-} (Y_i) f_i \quad (12)$$

In equations (11) and (12) m_+ and m_- is number of samples of class C^+ and C^- with f_i frequency of i^{th} sample in feature space (Y_i). The radius of C^+ ($Y_i \in C^+$) and C^- ($Y_i \in C^-$) with $n = \sum_i f_i$:

$$rd_+ = \frac{1}{n} \max \|\Phi_+ - \Phi(Y_i)\| \quad (13)$$

$$rd_- = \frac{1}{n} \max \|\Phi_- - \Phi(Y_i)\| \quad (14)$$

Then equation (13) can be written as equation (15) which is given in Appendix A. In equation (15) $Y' \in C^+$ and m_+ is number of training samples in C_+. Similarly, we have equation (16) as given in Appendix A.

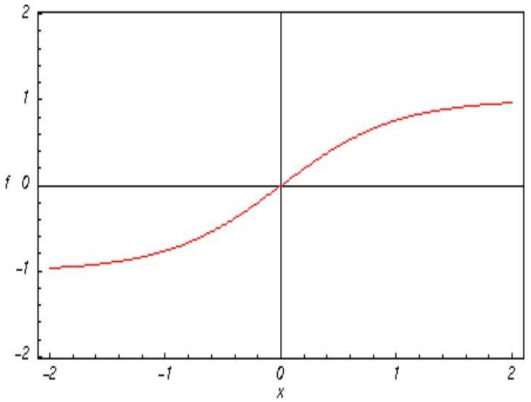

Figure 2: Hyperbolic Tangent Kernel.

In equation (16) $Y' \in C^-$ and m_- is number of training samples in C_-. The square of distance between sample $Y_i \in C^+$and $Y_i \in C^-$ to their class centres in feature space is:

$$dist_{i+}^2 = \|(Y_i) - \Phi_+\|^2 = \Phi^2(Y_i) - 2tanh\left[\Phi(Y_i) \cdot \Phi_+\right] + \Phi_+^2$$

$$dist_{i+}^2 = K(Y_i, Y_j) - \frac{2}{m_+} \sum_{Y_j \in C^+} K(Y_i, Y_j) + \frac{1}{m_+^2} \sum_{Y_j \in C^+} \sum_{Y_k \in C^+} K(Y_j, Y_k) \quad (17)$$

$$dist_{-}^{2}=K\left(Y_i,Y_j\right)-\frac{2}{m_{-}}\sum_{Y_j\in C^{-}}K\left(Y_i,Y_j\right)+$$

$$\frac{1}{m_{-}^{2}}\sum_{Y_j\in C^{-}}\sum_{Y_k\in C^{-}}K(Y_j,Y_k)\quad(18)$$

Now, $\forall i; i = 1,\ldots\ldots,M$ fuzzy rough membership function $frm_i(p)$ is defined in equation (19) as given in Appendix A. The term $(\cdot)$ in equation (19) holds when $(\exists i)\,\mu_{FC_i}(p) > 0$ and $\varepsilon > 0$ so that $frm_i(p) \neq 0$. Here, $\tau_{C_c}^{i} = \frac{\|FC_i\cap C_c\|}{\|FC_i\|}$ and $\frac{1}{\sum_i \mu_{FC_i}(p)}$ normalizes fuzzy rough membership function $\mu_{FC_i}(p)$. The function is a constrained fuzzy rough membership function. The above definition can further be modified as equation (20) which is given in Appendix A.

In equation (20) $\hat{H}$ is number of clusters and p has a non-zero membership. When p does not belong to any cluster then $\hat{H} = 0$ so that $\frac{\sum_{i=1}^{H}\mu_{FC_i}(p)\tau_{C_c}^{i}}{\hat{H}}$ becomes undefined. This issue is resolved by taking $frm_i^c(p) = 0$ when p does not belong to any cluster. This definition does not normalize fuzzy rough membership values and so the function is a possibilistic fuzzy rough membership function. The equations (19) and (20) expresses the fact that if an input pattern belongs to clusters (all belonging to only one class) with non-zero memberships then no fuzzy roughness are involved. However, in equation (20) it matters to what extent the pattern belongs to clusters. This is evident from property 11. Some of the important properties applicable to equations (19) and (20) are:

Property 1: $0<frm_i(p)<1$ and $0<frm_i^c(p)<1$

Property 2: $frm_i(p)/frm_i^c(p) = 1$ or 0 iff no uncertainty exists

Property 3: If no uncertainty is with p then $frm_i(p)/frm_i^c(p) = \tau_{C_c}^{i}$ for some $j \in \{1,2,\ldots\ldots,H\}$

Property 4: If no uncertainties are with p then $frm_i(p)/frm_i^c(p) = rm_A(p)$

Property 5: When each class is crisp and fine and class memberships are crisp $frm_i(p)/frm_i^c(p)$ is equivalent to fuzzy membership of p in class C_c

Property 6: If a and b are two patterns with $\mu_{FC_j}(a) = \mu_{FC_j}(b)\,\forall j$ and $\mu_{FC_{C_c}}(a) = \mu_{FC_{C_c}}(b)$ then $frm_i(a) = frm_i(b)$ and $frm_i^c(a) = frm_i^c(b)$

Property 7: $rm_{P-C_c}^{c}(p) = \left(\frac{\sum_{i=1}^{H}\mu_{FC_i}(p)\tau_{C_c}^{i}}{\hat{H}}\right) - rm_{C_c}^{c}(p) \wedge rm_{P-C_c}(p) = 1 - rm_{C_c}(p)$

Property 8: $\tau_{C_c\cup V}(p) \geq max\{\tau_{C_c}(p),\tau_V(p)\}$

Property 9: $\tau_{C_c\cap V}(p) \leq min\{\tau_{C_c}(p),\tau_V(p)\}$

Property 10: If W is family of pairwise disjoint crisp subsets of P then $\tau_{\cup W}(p) = \sum_{C_c\in W}\tau_{C_c}(p)$

Property 11: For C class classification problem with crisp classes, possibilistic fuzzy rough functions behave in possibilistic manner and constrained fuzzy rough functions behave otherwise

Property 12: If class is fuzzy then $0 \leq \sum_{c=1}^{C}\tau_{C_c}(p) \leq C$

The fuzzy rough membership values depend on fuzzy classification of input dataset. The fuzziness in classes represents fuzzy linguistic uncertainty present in dataset. The classification can be performed through either (a) unsupervised classification which involves collecting data from all classes and classify them subsequently without considering associated class labels with data or (b) supervised classification where separate datasets are formed for each class and classification is performed on each such dataset to find subgroups present in data from same class. Both classification tasks can be performed by some trivial classification algorithms [17], [18], [19]. However, there are certain problems which are to be taken care of such as: (a) number of classes which have to be fixed apriori or which may not be known (b) it will not work in case number of class is one and (c) generated fuzzy memberships are not possibilistic.

To overcome the first problem evolutionary programming based method may be used [18]. For various classification problems evolutionary methods can automatically determine number of classes. It is worth mentioning that number of classes should be determined as best as possible. Otherwise, calculation of fuzzy linguistic variables will be different and as a result fuzzy rough membership values may also vary. For the second problem if it is known apriori that only one class is present then mean and standard deviation are calculated from input dataset and π fuzzy membership curve is fitted. But while doing so care must be taken to detect possible presence of the outliers in input dataset. To overcome third problem possibilistic fuzzy classification algorithm or any mixed classification algorithm can be used. As of now there is no single classification algorithm which can solve all the problems. If output class is fuzzy then it may be possible to assign fuzzy memberships for output class subjectively. However, if domain specific knowledge is absent then we have to be satisfied with given crisp membership values.

The fuzzy rough ambiguity plays a critical role in many classification problems because of its capability towards modeling non statistical uncertainty. The characterization and quantification of fuzzy roughness are important aspects affecting management of uncertainty in classifier design. Hence measures of fuzzy roughness are essential to estimate average ambiguity in output class. A measure of fuzzy roughness for discrete output class $C_c \subseteq X$ is a mapping $S(X) \rightarrow \Re^{+}$ that quantifies degree of fuzzy roughness present in C_c. Here $S(X)$ is set of all fuzzy rough power sets defined within universal set X. The fuzzy rough ambiguity must be zero when there is no ambiguity in deciding whether an input pattern belongs to it or not. The equivalent classes form fuzzy classes so that each class is fuzzy linguistic variable. The membership is function of center and radius of each class in feature space and is represented with kernel.

In formulation of FRSVM, fuzzy membership reduces outliers' effects [18], [19]. When samples are nonlinear separable fuzzy memberships are calculated in input space but not in feature space. The contribution of each point in

hyperplane in feature space cannot be represented properly and fuzzy rough membership function efficiently solves this. Through fuzzy rough membership function the input is mapped into feature space. The fuzzy rough memberships are calculated in feature space. Further using kernel function it is not required to know shape of mapping function. This method represents contribution of each sample point towards separating hyperplane in feature space [19]. Thus, the proposed machine reduces outlier' effects efficiently and has better generalization ability.

The higher value of fuzzy rough membership function implies importance of data point to discriminate between classes. It implies highest value is given by support vectors. These vectors are training points which are not classified with confidence. These are examples whose corresponding $\alpha_i$values are non zero. From representer theorem [20] optimal weight vector w^* is linear combination of support vectors which are essential training points. The number n_{SV} of support vectors also characterizes complexity of learning task. If $n_{SV}$is small then only a few examples are important and rest can be disregarded. If n_{SV} is large then nearly every example is important for accuracy. It has been shown that under general assumptions about loss function and underlying distribution training data $n_{SV} = \Omega(n)$. This suggests that asymptotically all points are critical for training. While this gives $\Omega(n)$ bound on training time this solves FRSVM problem exactly. Further, datasets need not necessarily have $\Theta(n)$ support vectors.

6 MapReduce

MapReduce [21] illustrated in figure 3 is a programming model for processing large datasets with parallel distributed algorithm on cluster. It is composed of map and reduce function combinations derived from functional programming. The users specify map function that processes key value pair to generate a set of intermediate key value pairs and reduce function that merges all intermediate values associated with same intermediate key [21]. MapReduce is divided into two major phases called map and reduce separated by an internal shuffle phase of intermediate results. The framework automatically executes those functions in parallel over n number of processors. MapReduce job executes three basic operations on dataset distributed across many shared nothing cluster nodes. The first task is map function that is processed in parallel manner by each node without transferring any data with other nodes. In next operation processed data by map function is repartitioned across all nodes of cluster. Finally reduce task is executed in parallel by each node with partitioned data.

A file in distributed file system is split into multiple chunks and each chunk is stored on different data nodes. A map function takes key value pair as input from input chunks and produces list of key value pairs as output. The type of output key and value can be different from input values.

$$map(key_1, value_1) \Rightarrow list(key_2, value_2) \quad (21)$$

A reduce function takes key and associated value list as input and generates list of new values as output:

$$reduce(key_2, list(value_2)) \Rightarrow list(value_3) \quad (22)$$

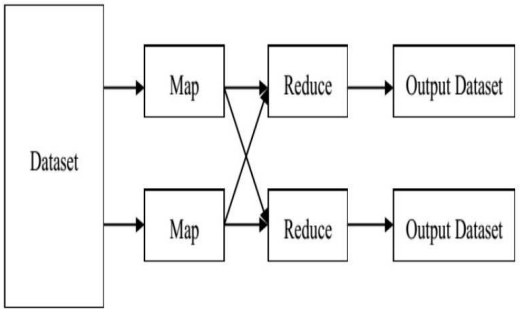

Figure 3: MapReduce System.

Each reduce call produces either value $value_3$ or an empty return, through one call returns more than one value. The return of all calls is collected as desired result list. The main advantage of MapReduce is that it allows distributed processing of submitted job on subset of whole dataset in the network.

7 Experimental framework: Parallel fuzzy rough support vector machine

FRSVM suffers from the scalability problem [22] both in terms of memory and computational time. In order to improve the scalability problem a parallel FRSVM viz. PFRSVM is developed which handles the stated problems through parallel computation. It is executed through multiple commodity computing nodes on cascade FRSVM model parallel in cloud environment. PFRSVM training is realized through FRSVMs where each FRSVM acts as filter. This leads to the process of deriving local optimal solutions which contribute towards the global optimum solution. Through PFRSVM huge scale data optimization problems are divided into independent small optimization problems. The support vectors of the prior FRSVM are used as the input of later FRSVM. FRSVM is aggregated into PFRSVM in hierarchical fashion. The PFRSVM training process is described in the figure 4.

In this architecture, support vectors sets of two FRSVMs are combined together as input to new FRSVM. This process continues until only one vector set remains. Here a single FRSVM never deals with the whole training set. If filters in the first few layers are efficient in extracting more

support vectors it leads to maximum optimization. This results in handling fewer support vectors in the later layers. Thus training sets of each of the sub problems are much smaller than that of whole problem where support vectors are a small subset of training vectors. For training FRSVM classifier functions LibSVM [23] with various kernels. The cross validation test is used to find appropriate values of parameters C and γ as discussed in section 8. The entire framework is implemented with Hadoop and streaming Python package.

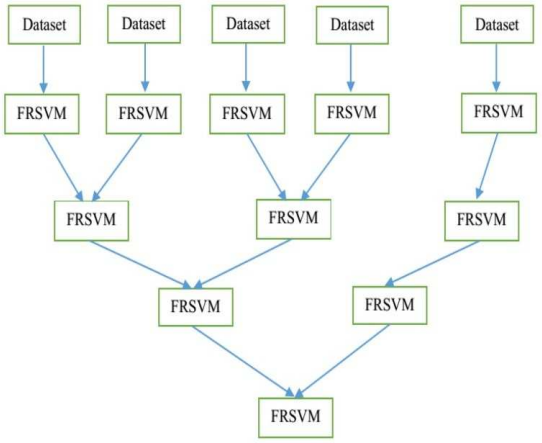

Figure 4: The training flow of PFRSVM.

Given computing nodes in cloud environment the original large scale data SD is partitioned into smaller data sections $\{SD_1, \ldots\ldots, SD_n\}$ uniformly. These small data sets SD_i are placed on the computing nodes. Then the corresponding partition files are created. Based on the available computation environment the job configuration manager [24] configures the computation parameters such as map, reduce, class names combination, number of map and reduce tasks, partition file etc. The driver manager [24] initiates the MapReduce task. The dynamic parameters are transformed to each computing node through an API interface [24].

In each computing node the map tasks are operated. The first layer of figure 4 loads the sample data from local file system according to the partition file. Each node in the layer classifies partitioned dataset SD_i locally through FRSVM from which support vectors are obtained. In following layers training samples are support vectors of the former layer. The local support vectors obtained in earlier layers are merged with global support vectors in the later layers. LibSVM is used to train each FRSVM. In LibSVM sequential minimal optimization [23] is used to select workset in decomposition methods for training FRSVM. FRSVM is trained using [23].

In map job of MapReduce subset of training set is combined with other local support vectors. The trained support vectors are sent to reduce jobs. In reduce job support vectors of all map jobs are collected, evaluated, merged with global support vectors and fed back to the client. Each computer within cloud environment reads global support vectors. Then it merges global support vectors with subsets of local training data and classifies via FRSVM. Finally, all computed support vectors in cloud computers are merged. The algorithm saves global support vectors with new ones. The training process is performed iteratively and stops when all FRSVMs are combined together resulting into PFRSVM. The entire system is schematically shown in figure 5. The steps of algorithm are:

PFRSVM Algorithm

1. Initialize global support vector set $\left[i = 0, GV^i = \phi\right]$
2. $i = i + 1$
3. For any computing node $c \in C$
   Read global support vectors
   Merge them with subset of training data
4. Train FRSVM with merged new dataset
5. Find support vectors
6. When all computers in cloud complete training
   Merge all calculated support vectors
   Save to global support vector set
7. If $(t^i = t^{i-1})$
   Stop
   else
   Goto 2

The map and reduce functions of PFRSVM are:

Map function of PFRSVM Algorithm

GV= ϕ
While $(t^i \neq t^{i-1})$ do
for $c \in C$ do

$$TS_c^i = TS_c^i \cup GV^i$$

end for
end while

Reduce function of PFRSVM Algorithm

While $(t^i \neq t^{i-1})$ do
for $c \in C$ do

$$SV_c, t^i = frsvm(TS_c)$$

end for
for $c \in C$ do

$$GV = GV \cup SV_C$$

end for
end while

The architecture of PFRSVM developed is not able to achieve linear speedup when number of machines continues to increase beyond a data size dependent threshold. This happens because of communication and synchronization overheads between the computing nodes. The communication overhead occurs when message passing takes place between machines. The synchronization overhead occurs when master node waits for task completion on slowest machine. The MapReduce compatible algorithm runs with Hadoop cluster [25] which uses identical software versions and hardware configurations through which linear speedup is achieved.

Another aspect which deserves attention is convergence of PFRSVM while performing classification [12], [19]. To consider this let us assume a subset ST of training set TS with $OPT\left(ST\right)$ as optimal objective function over ST. Here H^* is global optimal hypothesis with minimal empirical risk $RK_{emp}\left(H^*\right)$. The algorithm starts with $GV^0 = 0$ and generates a decreasing sequence of positive set of vectors GV^i with following hinge loss function:

$$HL\left(f\left(x\right),y\right) = max\left[0, 1 - y \cdot f\left(x\right)\right] \quad (23)$$

The empirical risk is computed with approximation:

$$RK_{emp}\left(H\right) = \frac{1}{n}\sum_{i=1}^{n} HL\left(H\left(x_i\right), y_i\right) \quad (24)$$

According to empirical risk minimization principle learning algorithm chooses hypothesis $\hat{H}$ minimizing risk:

$$\hat{H} = arg\ min_{H\in\mathcal{H}}\ RK_{emp}\left(H\right) \quad (25)$$

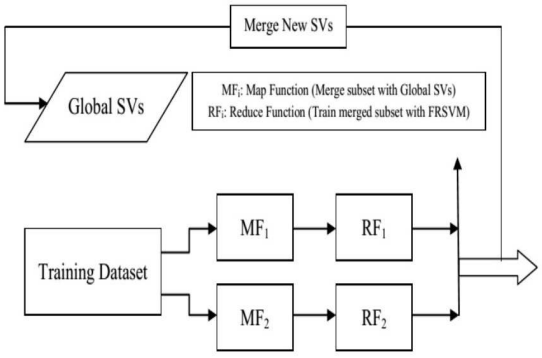

Figure 5: Schematic representation of PFRSVM in Cloud Environment.

A hypothesis exists in every cloud node. Let Y be subset of training data at cloud node j; $H^{i,j}$ is hypothesis at node j with iteration i such that optimization problem in equation (3) and corresponding equation (10) becomes equation (26) given in Appendix A.

In equation (26) KM_{12} and KM_{21} are kernel matrices with respect to $KM_{12} = \left\{K_{j,k}\left(y_{jk}, GV^i_{(j,k)}\right)|j = 1,..,m; k = 1,..n\right\}$. Here α_1 and α_2 are solutions estimated by node j with dataset Y and GV. The kernel matrix KM is symmetric positive definite on square because of Mercer's condition as a result of which KM_{12} and KM_{21} are equal. At iteration i matrices $KM_{11} = \left\{K_{j,k}\left(y_{jk}, y_{jk}\right)|y_{jk} \in Y, j = 1,..,m; k = 1,..n\right\}$ and

$$KM_{22} = \left\{K_{j,k}\left(GV, GV\right)|j = 1,..,m; k = 1,..n\right\}.$$

The algorithm terminates when hypothesis' empirical risk is same with previous iteration i.e. $RK_{emp}\left(H^i\right) = RK_{emp}\left(H^{i-1}\right)$. The accuracy of decision function of PFRSVM classifier at i^{th} iteration is always greater than or equal to maximum accuracy of decision function of SVM classifier at 1^{st}iteration i.e. $RK_{emp}\left(H^i\right) \leq arg\ min_{H\in\mathcal{H}^{i-1}}\ RK_{emp}\left(H\right)$.

Finally we discuss the complexity of proposed algorithm. The time complexity of FRSVM is $O\left(n^2\right)$. The bandwidth of network determines the transmission time of data T_{tt}between map and reduce nodes. When training data is divided into p partitions computation cost is calculated in terms of layers of cascade FRSVM as $\log_2 p$. Considering ratio between number of support vectors and whole training sample as $a\left(0 < a < 1\right)$ and ratio between support vectors and training sample excluding first layer as $b(1 < b < 2)$, the number of training samples of i^{th} layer is $nab\left(\frac{b}{2}\right)^{\log_2 p\ -i}$. The computation time is: $O\left(\left(\frac{n}{p}\right)^2\right) + \sum_i O\left(\left(nab\left(\frac{b}{2}\right)^{\log_2 p\ -i}\right)^2\right) + O\left(\sum_i nab\left(\frac{b}{2}\right)^{\log_2 p\ -i-1} 2^{\log_2 p\ -i-1}\right) + T_{tt}$. The overhead of data transfer includes three parts: (a) the first part is data transfer from map to reduce nodes which are support vectors obtained by map nodes (b) the second part is data transfer from reduce to server node which are support vectors and (c) the third part is data transfer from server nodes to map node which are training samples combined by support vectors. The overhead of data transfer depends on bandwidth of MapReduce cluster.

8 Experimental results and discussions

In this section, the effectiveness of PFRSVM is demonstrated with various experiments. At first a brief discussion on generation of synthetic data is given. Then real datasets used are highlighted. Next we illustrate selection of optimum values of (C, γ) and kernel used in PFRSVM. The classification on synthetic data follows this. Next the outlier generation in real data is given. This is followed by classification on real data. The imbalance and overlapping class classification is presented next. This is followed by generalization to unseen data when size of training and test dataset are varied. The discussion on features and labels relaxation follows this. The comparative classification performance of PFRSVM with other approaches is given next. Finally, some critical issues regarding implementation of

PFRSVM in cloud environment is highlighted.

8.1 Generation of synthetic data

To validate performance of PFRSVM in realistic environments the datasets are generated as: (a) We randomly created S clusters such that for each cluster: (i) centre $cp \in [cp_l, cp_h]$ for each dimension independently; (ii) radius $rd \in [rd_l, rd_h]$ and (iii) number of points $np \in [np_l, np_h]$ in each cluster (b) We labeled clusters based on X-axis value such that cluster T_i is labeled as positive if $cp_i^x < \alpha - rd_i$ and negative if $cp_i^x > \alpha + rd_i$. Here $cp_i^x$is X-axis of centre cp_i and α is threshold between $[cp_l, cp_h]$. We removed clusters not assigned to either of positive or negative which lie across threshold α on X-axis to make them linearly separable. (c) Once characteristics of each cluster are determined points for clusters are generated according to 2-dimensional independent normal distribution with $[cp, rd]$ as mean and standard deviation. The class label of each data is inherited from label of its parent cluster. It is noted that due to normal distribution maximum distance between a point in cluster and centre is unbounded. The points that belong to one cluster but located farther than surface of cluster are treated as outsiders. Due to this dataset does not become completely linearly separable. Figure 6 shows a dataset according to parameters (see Table 1). The data generated from clusters in left and right side are positive ('l') and negative ('-') respectively. Figure 6(b) shows 0.5 % randomly sampled data from original dataset of figure 6(a). From figure 6(b) the random sampling reflects unstable data distribution of original dataset which includes nontrivial amount of unnecessary points. The dashed ellipses on Figure 6(b) indicate densely sampled areas that reflect original data distribution are mostly not very close to boundary. As such areas around boundary are less dense because cluster centers which are very dense are unlikely to cross over boundary of multiple classes. Thus unnecessary data increases training time of PFRSVM without contributing to support vectors of boundary. The random sampling disturbs more when probability distributions of training and testing data are different because random sampling only reflects that distribution of training data could miss significant regions of testing data. This happens as they are collected in different time periods. For fair evaluation testing data is generated using same clusters and radiuses but different probability distributions by randomly reassigning number of points for each cluster.

8.2 Real datasets used

The real datasets from UCI Machine Learning Repository viz. German Credit, Heart Disease, Ionosphere, Semeion Handwritten Digit and Landsat Satellite are used to conduct experiments and illustrate convergence of PFRSVM. The different attributes of datasets are given (see Table 2 in Appendix B). The missing values' problem in datasets is resolved by genetic programming [19]. The nominal values

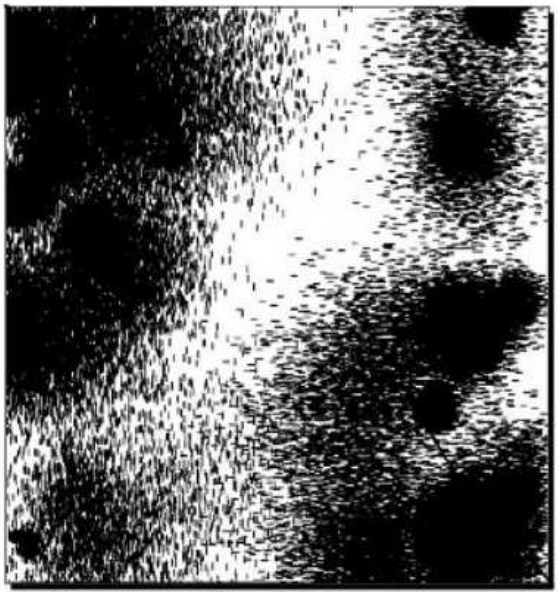

Figure 6: Original dataset $[N = 114996]$.

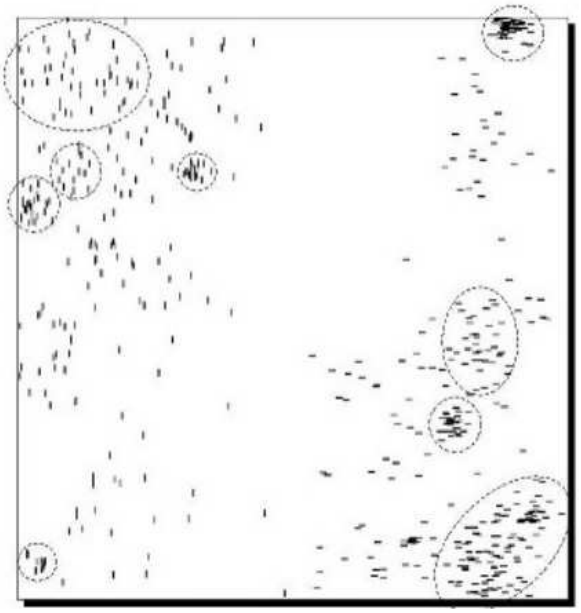

Figure 7: 0.5 % randomly sampled data $[N = 602]$

Figure 8: Synthetic Dataset in 2-Dimensional Space.

are converted into numerical ones during processing. The datasets are divided into training and test sets each consisting of $50\ \%$ samples. The training set is created by randomly selecting samples from whole dataset and remaining samples constitute test set.

Parameter	Values
Number of clusters S	70
Range of cp $[cp_l, cp_h]$	$[0.0,\ 1.0]$
Range of rd $[rd_l, rd_h]$	$[0.0,\ 0.1]$
Range of np $[np_l, np_h]$	$[0,\ 5000]$
α	0.7

Table 1: Data Generated for figure 6.

8.3 Selection of optimum values of (C, γ) and kernel used in PFRSVM

Selection of appropriate PFRSVM parameters plays a vital role in achieving good results. We consider RBF kernel and use cross validation to find best parameters of (C, γ) to train and test whole training set. A common strategy is to separate dataset into known and unknown parts. The prediction accuracy obtained from unknown set precisely reflects classifying performance on an independent dataset. An improved version viz. v fold cross validation is used $(v = 20)$ where training set is divided into v equal subsets. One subset is tested using classifier trained on remaining $(v - 1)$ subsets. Each instance of whole training set is predicted once cross validation accuracy is data percentage correctly classified. This prevents over fitting problem. The grid search also finds (C, γ) using cross validation. Various pairs of (C, γ) values are tried and best cross validation accuracy is selected. We found that exponentially growing sequences of (C, γ) viz. $(C = 2^{-5}, 2^{-3}, \ldots\ldots, 2^{15}; \gamma = 2^{-15}, 2^{-13}, \ldots\ldots, 2^{3})$ give best results. The grid search performs exhaustive parameter search by using heuristics with good complexity. The kernel used here is RBF. It has been used with considerable success so its choice is obvious. It nonlinearly maps samples into higher dimensional space when relation between class labels and attributes is nonlinear. RBF kernel has less hyper parameters which also influences complexity of model selection. Also RBF kernel has fewer numerical difficulties. After scaling the datasets we first use grid search and find average best (C, γ) values as $(2^3, 2^{-7.37})$ with average cross validation rate $83\ \%$. After the best (C, γ) is found whole training set is trained to generate final classifier. The proposed approach works well with thousands or more points.

8.4 Classification on synthetic data

Considering the synthetic data generated in section 8.1 Table 3 in Appendix B shows results on testing dataset. PFRVM accuracy is evaluated for different values of $C \in \{0.5,\ 1,\ 1.5,\ 2,\ 5,\ 10,\ 20\}$. The best value is when $C = 20$. For larger C PFRSVM accuracy improves and error decreases. The number of false predictions is reported on testing dataset because of data size. PFRSVM outperforms SVM with same number of random samples. The FRSVM training time is almost $0.5\ \%$ of random samples. The sampling time for PFRSVM constructs 572 data points. With the growth of data size random sample gives similar accuracies as PFRSVM. The training time of SVM with random sampling is longer than PFRSVM. It is evident that using standard kernel functions good classification results are produced.

A larger dataset is generated according to parameters to verify PFRSVM performance (see Table 3 in Appendix B). The results of random sampling, MFSVM and PFRSVM on large dataset are also given (see Table 4 in Appendix B). Due to simple linear boundary on large training data random sampling does not increase MFSVM performance when sample size grows. The error rates of MFSVM and PFRSVM are approximately around $15\ \%$ lower than random sampling of highest performance. The total training time of PFRSVM including sampling time is less than MFSVM or random sampling of highest performance. For voluminous datasets MFSVM takes longer time than PFRSVM. MFSVM is executed with $\delta = 7$ starting from one positive and one negative sample and adding seven samples at each round yielding good results. The value of δ is set below 10. If δ is too high, its performance converges slower with larger amount of training data to achieve same accuracy. If δ is too low, MFSVM may need to undergo too many rounds.

8.5 Generation of outliers in real data

The outliers are generated from real datasets using distance based outliers algorithm [19]. Each point is ranked on basis of its distance to k^{th} nearest neighbor and top n points are declared as outliers. Also classical nested loop join and index join partition based algorithms are used for mining outliers. The input dataset are first partitioned into disjoint subsets. The entire partitions are pruned when they cannot contain outliers resulting in substantial savings in computation. The partition based algorithm scales well with respect to both dataset size and dimensionality. The performance results are dependent on number of points, k^{th} nearest neighbor, number of outliers and dimensions [26].

8.6 Classification on real data

Now we consider real datasets and study the comparative performance of PFRSVM with FRSVM. The experimental results using Gaussian RBF and Bayesian kernels are listed [17], [18] (see Tables 5 and 6 in Appendix B). Both training and testing rates are highlighted. For different datasets the values of C considered are 8 and 128. When Gaussian kernel is used PFRSVM achieve highest test rate. When bayesian kernel is used then also PFRSVM have better generation performance for Ionosphere and Semeion

Handwritten Digit datasets. The table also illustrates that PFRSVM has better generalization ability than FRSVM. Finally PFRSVM has better performance than FRSVM on reducing outliers' effects.

8.7 Classification with imbalance and overlapping classes

PFRSVM resolves the class overlapping combined with imbalance problem effectively. It is different from traditional classifiers using crisp decision producing high misclassifications rates. The soft decision of PFRSVM with optimized overlapping region detection addresses this. It provides multiple decision options. The overlapping region detected optimizes performance index that balances classification accuracy of crisp and cost of soft decisions. The optimized overlapping regions divide feature space into two parts with low and high confidence of correct classification. For test data falling into overlapping regions multiple decision options and measures of confidence are produced for analysis while for test data falling into non overlapping regions results in crisp decisions. The training procedure first builds fuzzy rough soft model using training data. The fuzzy rough information of training data $FR_{training}$ is used to search optimal threshold θ^* defining overlapping region. In testing stage incoming data is first processed to find its location. In feature space X overlapping region $R(\theta)$ is centered at decision boundary with margin at each boundary side. The width of margin is determined by threshold θ as given by Equation(27) in Appendix A. Here $FR(\omega_i|X)$ is posterior fuzzy rough measure of class i given X. The location of decision boundary $(FR(\omega_1|X) = FR(\omega_2|X))$ is determined by class distribution of training data. In overlapping region detection the problem is determination of θ. In overlapping region detection two considerations should be taken (i) region should be large enough to cover most potentially misclassified data so that classification in non-overlapping region is highly accurate and (ii) region should be compact so as to avoid making soft decisions to too many patterns as patterns with soft decisions are verified by system and hence increase cost. To implement the stated considerations two criteria i.e. classification accuracy in non-overlapping regions $acc(\theta)$ and cost $c(\theta)$ are considered. To find a good tradeoff between $acc(\theta)$ and $c(\theta)$ an aggregate performance evaluation criterion is achieved through weighted harmonic mean $HM_\beta(\theta)$. The weight parameter β is predefined to attend to accuracy in non-overlapping region $\frac{\beta}{1-\beta}$ times as volume of non overlapping region. The default $\beta = 0.86$ since decreasing rate of $c(\theta)$ is always faster than increasing rate of $acc(\theta)$. The optimal threshold θ^* maximizes criterion $HM_\beta(\theta)$. By using this optimization criterion optimal volume of overlapping region is able to adapt to various data distributions and overlapping degrees. This criterion can be extended to multiple overlapping classes.

8.8 Generalization to unseen data when varying the size of training to test dataset

PFRSVM generalizes well to unseen data when size of training to test dataset is varied. It has been observed that dataset sizes have been growing steadily larger over the years. This leads development of training algorithms that scale at worst linearly with number of examples. Supervised learning involves analyzing given set of labeled observations (training set) so as to predict labels of unlabeled future data (test set). Specifically, the goal is to learn some function that describes relationship between observations and their labels. The interest parameter here is size of training set. The learning problem is called large scale if its training set cannot be stored in memory [25]. In large scale learning main computational constraint is time available rather than number of examples. In order for algorithms to be feasible on datasets they scale at worst linearly with number of examples.

The dual quadratic programming method in PFRSVM favors smooth handing of kernels. With focus of problem in large scale setting several methods have shifted back to primal. But dual is amenable to certain optimization techniques that converge quickly. The dual solvers used are special techniques to quickly achieve good solution. There are exponentially many constraints to problem which are expected for structural prediction. It is desirable that there is a single slack variable shared across each of these constraints. This affords some flexibility in solving the problem. The problem is solved through cutting plane method. The problem is approached iteratively by keeping working set W of constraints and restricted to constraints in W. Each element of W is vector $w \in \{0,\ 1\}^n$ and is considered as some combination of training point indices. The working set is updated each iteration to include indices for points that are currently misclassified. The algorithm terminates when it is within optimal primal solution and it achieves with appreciable time. However, training time increases such that it takes longer to reach an approximate solution.

8.9 Relaxation of dependency between features and labels

PFRSVM effectively relaxes dependencies between features of an element and its label. In this direction sequence labeling is used which identifies best assignment to collection of features so that it is consistent with dependencies set. The dependencies constrain output space. The dependencies are modeled with constraints so that it is a constrained assignment problem. To solve this, two-step process [27] is used that relies on constraint satisfaction algorithm viz. relaxation labeling. First PFRSVM classifier affects initial assignment to features without considering dependencies and then relaxation process applies successively to constraints to propagate information and ensure

global consistency. It aims at estimating for each feature probability distribution over labels set. To produce these maximum entropy framework is adopted. It models joint distribution of labels and input features. The probability of labeling feature tr with label λ is modeled as exponential distribution:

$$prob\left(\lambda|tr;\theta\right) = \frac{1}{Z_\theta\left(tr\right)} exp\left\langle\theta, \phi\left(tr,\lambda\right)\right\rangle \quad (28)$$

Here $\phi\left(tr,\lambda\right)$ is feature vector describing jointly feature tr and label λ; $Z_\theta\left(tr\right)$ is normalizing factor and θ is parameter vector. To estimate θ maximum entropy framework advocates among all probability distributions that satisfy constraints imposed by training set the one with highest entropy. Relaxation labeling is an iterative optimization technique that solves efficiently assigning labels problem to features set satisfying constraints set. It reaches an assignment with maximal consensus among labels and feature sets. Denoting $TR = \{tr_1, \ldots.., tr_n\}$ set of n features, Λ is set of m possible labels and λ and ????? two features of Λ. The interactions between labels are denoted by compatibility matrix $CM = \{cm_{ij}\left(\lambda,\mu\right)\}$. The coefficient $cm_{ij}\left(\lambda,\mu\right)$ represents constraint and measures to which extent i^{th} feature is modeled with label λ when label of j^{th} feature is μ. These coefficients are estimated from training set. The algorithm starts from an initial label assignment from classifier. The relaxation iteratively modifies this assignment so that labeling globally satisfies constraints described by compatibility matrix. All labels are updated in parallel using information provided by compatibility matrix and current label assignment. For each feature tr_i and each label λ support function describes compatibility of hypothesis label of tr_i is λ and current label assignment of other features defined by:

$$q_i^{(t)}\left(\lambda;\overline{p}\right) = \sum_{j=1}^{n}\sum_{\mu\in\Lambda} cm_{ij}\left(\lambda,\mu\right)p_j^{(t)}\left(\mu\right) \quad (29)$$

In equation (29) $\overline{p} = \{\overline{p}_1, \ldots.., \overline{p}_n\}$ is weighted label assignment and each $p_j^{(t)}\left(\mu\right)$ in current confidence in hypothesis label of i^{th} feature is λ. The weighted assignment is updated to increase $p_i\left(\lambda\right)$ when $q_i\left(\lambda\right)$ is big and decrease it otherwise. More precisely update of each $p_i\left(\lambda\right)$ is:

$$p_i^{(t+1)}\left(\lambda\right) \leftarrow \frac{p_i^{(t)}\left(\lambda\right)\cdot q_i^{(t)}\left(\lambda,\overline{p}^{(t)}\right)}{\sum_{\mu\in\Lambda} p_i^{(t)}\left(\lambda\right)\cdot q_i^{(t)}\left(\lambda,\overline{p}^{(t)}\right)} \quad (30)$$

The calculation of support and mapping update is iterated until convergence.

8.10 Comparative classification performance of PFRSVM with other approaches

Keeping in view the results on real datasets a comparative average classification performance of PFRSVM is presented here with respect to parameters $C = 8, \gamma = 2^{-7.37}$ and Gaussian RBF and Bayesian kernels. The results are also highlighted (see Tables 7 and 8 in Appendix B). It is evident that PFRSVM achieves superior average classification accuracy percentage as compared to other SVM versions for both kernels.

8.11 Some critical issues regarding implementation of PFRSVM in cloud environment

PFRSVM implemented here presents a powerful method of huge datasets classification in distributed cloud environment. In this process we have discussed a new architecture of in-stream data analytics [24]. The parallel approach of computation offers a significant processing model for handling discontinuity in data thereby enabling the development of a scalable and reliable system [28]. The application processes real time streams of data which pushes the limits of traditional data processing architectures. This leads to a new class of system software viz. stream processing engine whose attributes are characterized by a core set of the following eight general rules [29]:

1. Always keep the data moving
2. Place stream based query from the database
3. Handle the stream imperfections through delayed, missing and out-of-order data
4. Always generate predictable outcomes
5. Effectively integrate the stored and streaming data
6. Always guarantee data safety and availability
7. Automatically partition and scale the applications
8. Always process and respond instantaneously

The architecture of PFRSVM as shown in figure 7 is developed keeping in view the above eight rules of stream processing [29]. The overall architecture of the system provides guaranteed order independence which is challenging and also vital in building scalable and reliable systems. In what follows we highlight various critical problems solved in order to build an enterprise class stream data classification engine [24]:

1. Architecture of stream processing engine: The architecture of the system is designed by splitting the entire computational workload into parallel phases that relies on partial processing. These partial phases are eventually combined together serially that operates using traditional approach. This entails identification of the portions of computation that are suitable for parallel processing, pushing partial workload processing right to the input manager of the data loading pipeline and hooking up the results of concurrent partial processing to serial processing [30]. The parallel

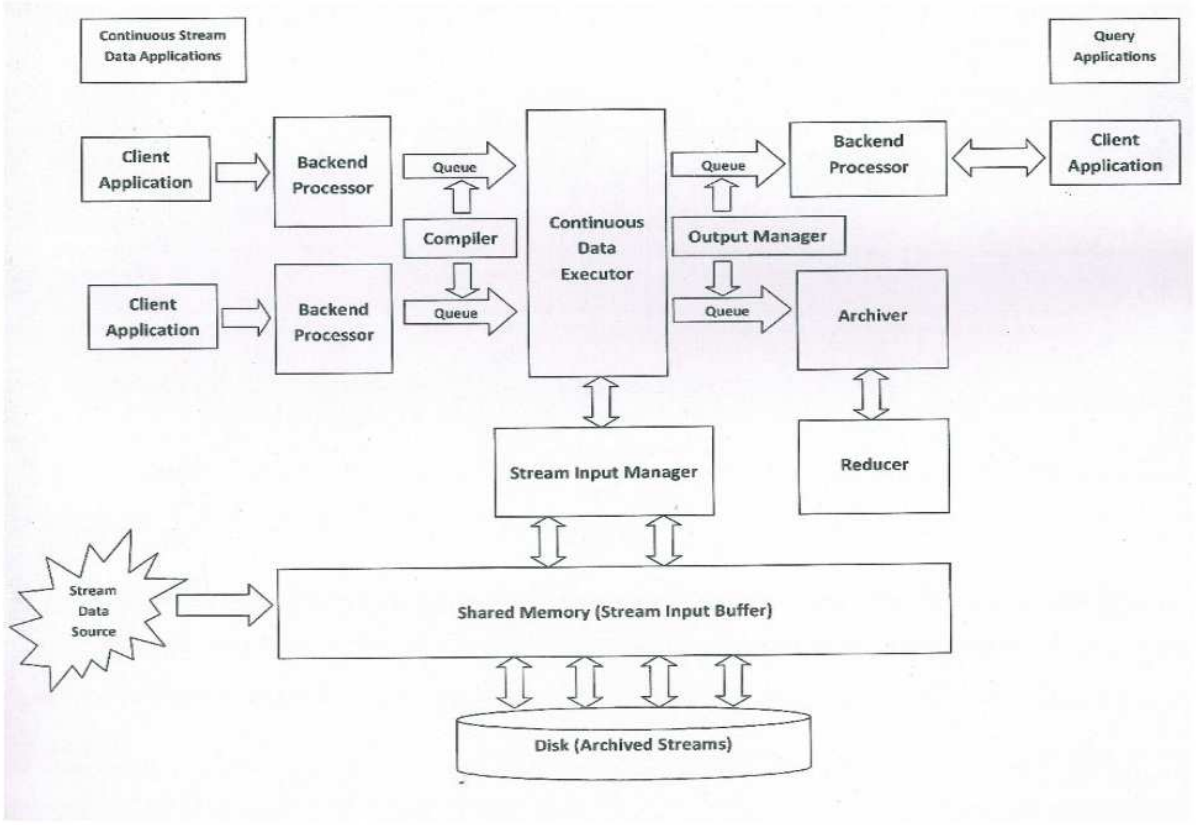

Figure 9: The architecture of PFRSVM.

processing streams handle raw and intermediate derived streams earmarked for parallel phase and serial processing streams handle derived streams and other computations that operate in serial phase. The data sources are connected to the system using standard protocols and start pumping data into streams using bulk loading. The system forks an associated thread dedicated to each connection. This thread instantiates a local data flow network of computation operators on appropriate stream on which data arrives. These operators are specifically generated for parallel processing streams that correspond to particular raw stream and produces result runs. The local data flow network is also responsible for archiving raw data into tables it processes as well as corresponding parallel processing results it produces. It also sends parallel processing data to the shared workload executor for use in processing serial processing streams via shared memory queues [24]. The executor receives similar threads that services other data sources. As such the executor is part of a single very long running transaction for its entire existence. The executor thread fetches parallel processing records from input queues and processes them through a data flow network. The executor exploits multiple available cores in the system by splitting operations across multiple threads. The system also includes an archiver thread responsible for writing out windows of data produced for serial processing streams by the executor to tables. There also exists a reducer thread responsible for eagerly combining partial results in background and a repair thread that continually repairs contents of archives of serial processing streams. It is always possible to spawn several instances each executor, archiver, reducer and repair threads.

2. Order independence in data processing: In this computational framework order independence is always achieved for parallel processing streams. For serial processing streams order independence is implemented by processing out of order data by periodically repairing any affected archived serial processing data. The archives are always correct with respect to out of order data on an eventual consistency basis. It involves two activities viz. (i) spooling all data tuples that arrive too late into an auxiliary structure through a system generated corrections table and (ii) a repair process that periodically scan records from auxiliary table and combines them with an appropriate portion of originally arrived tuples in order to recompute and update affected portions of archived serial processing data. This approach is similar to the dynamic revision

of results as illustrated in [31].

3. Streamlining computational transaction: The computations in PFRSVM are defined through transactions which define the unit of work [24]. The transaction is associated with well-known ACID properties viz. atomicity, consistency, isolation and durability. The focus here is basically on atomicity, consistency and durability. Atomicity is vital in order to easily undo the effects of failures in either individual computations or the entire system. Consistency leads to integrity of data processed in the system. Durability is critical in order to recover state of the system after a crash. These properties are key attributors in the way data is loaded into the in-stream analytic system where loading application batches up a dataset and loads it in a single unit of work. This model is vital in connecting a transactional message queue with streaming system such that no records can ever be lost. This depends on the ability to abort a data loading transaction either based on an error condition which may occur in the loader, network or system. On abort it is vital that all modifications to raw and derived stream histories must be rolled back at least eventually. It is very challenging to support the abort requirement here because waiting until data is committed before processing leads to significant extra latency defeats the purpose of the streaming system and processing dirty uncommitted data from multiple transactions makes it hard to unwind the effects of a single transaction. The latter is particularly hard because archiving of serial processing streams is the responsibility of archiver threads that run their own transactions and are independent of thread that manages the unit of work in which data is loaded. The solution is achieved through two stages where we push down the partial processing and archiving results to input manager thread that handles the data loading transaction and we organize the data loading application into several concurrent units of work each of which loads one or more chunks of data. The data chunk is a finite subpart of stream that arises naturally as by product, the way data is collected and sent to a streaming system. Here the individual systems spool data into log files kept locally. These files are bulk loaded through standard interfaces and are often split at convenient boundaries based on number of records, size or time and are sent separately to the stream processing engine. They also serve as natural units for data chunks with clear boundaries.

   Consider an example of order independent partial processing transaction where certain types of operations are implemented that are tolerant to appreciable amounts of disorder in their input [24]. The transaction adheres to all the ACID properties stated earlier. Let us take tumbling or non-overlapping window count query with window of 6 minutes shown in figure 8 which operates over a stream of data with many

select count(*), close(*) s from V ⟨slices '6 minutes'⟩

Row Number	Input		State Partitions			Output Tuples
	Data	Control	Part 1	Part 2	Part 3	
1	1		1			
2	3		2			
3	2		3			
4	4		4			
5	2		5			
6	1		6			
7		5				(6, 5)
8	6		1			
9	4		1	1		
10	3		2	1		
11	7		3	1		
12	3		3	2		
13		10		2		(3, 10)
14	12		1	2		
15	8		1	1		(4, 5)
16	6			1	1	
17	3			1	2	
18	9			2	2	
19		15		2	2	(1, 15)
20		flush 2			2	(4, 10)
21		flush 3				(4, 5)

The close (*) aggregate function returns timestamp at closure of relevant window

Figure 10: A transaction illustrating the Order Independent Partial Processing adhering ACID properties.

   records arriving out-of-order and input manager provides progress information on heuristic basis. This has resulted in 6 out-of-order tuples being discarded. Here we have 3 columns each representing the state of an individual partition. The first window returns the behavior of an out-of-order processing approach. The second window the arrival of an out-of-order tuple with timestamp 3 (row 10) reduces the system to second partition. When an out-of-order tuple with timestamp 3 arrives during that same window it is handled in second partition as it is still in-order relative to that partition. When tuple with timestamp 6 (in row 16) comes in during third window its timestamp is high enough to cause the open window of second partition to close producing partial result of (2, 5) and processing new tuple in second partition associated with second window ending at time 10. When next two tuples (rows 17 and 18) with timestamps 3 and 9 come in they are too late to be processed in second partition and requires the system to proceed to third partition where they are sent. Next tuple with timestamp 9 (row 18) comes in and is sent to second partition. When system receives a control tuple with timestamp 15 it flushes second and third partitions producing partial results of (2, 10) and (2, 5).

4. Recovery from failure: Once the system fails it is recovered by bringing the system back to a consistent state after a crash when all in flight transactions during the crash are deemed aborted [24]. Here the recovery archives of parallel processing streams are free since all writes of raw and corresponding derived data hap-

pen as part of the same transaction. We benefit both from the robust recovery architecture of the storage subsystem and also from other features such as online backup mechanisms. The recovery operation for serial processing is a more challenging because the large amounts of runtime state managed in main memory structures by operators in executor and the decoupled nature in which durable state is written out originally by the archiver. The crash recovery therefore involves standard database style recovery of all durable state, making all serial processing archives self-consistent with each other as well as the latest committed data from their underlying archive and rebuilding the runtime state of the various operators in executor. The ability to have robust recovery implementation that is capable of quickly recovering from a failure is essential. Furthermore, the longer it takes to recover from a failure the more the amount of pent-up data has gathered and the longer it takes to catch up to the live data.

The recovery from failure in distributed PFRSVM is assessed in terms of delay, process and correct (DPC) protocol which handles crash failures of processing nodes and network failures [24]. Here, the choice is made explicit as the user specifies an availability bound and it attempts to minimize the resulting inconsistency between the node replicas while meeting the given delay threshold. The protocol tolerates occurrence of multiple simultaneous failures that occur during recovery. In DPC each node replica manages its own availability and consistency by implementing state machine as shown in figure 9 that has three states viz. STABLE, UPSTREAM FAILURE (UP FAILURE), and STABILIZATION.

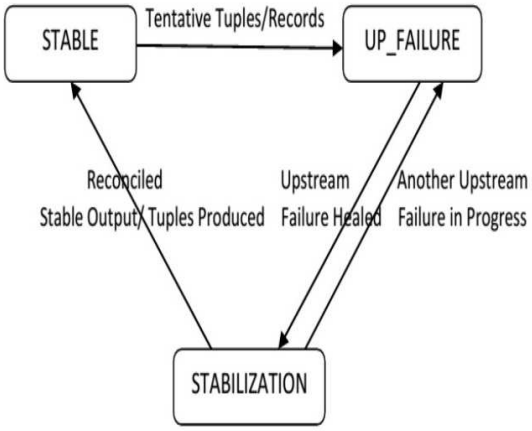

Figure 11: The DPC State Machine.

The DPC provides eventual consistency even when multiple failures overlap in time and with at least two replicas of any processing node it maintains the required availability at all times. In both scenarios client applications eventually receive stable version of all result tuples and there are no duplications. A single processing node with no replica, no upstream and downstream neighbors is executed and its inputs are controlled directly. The node produces complete and correct output stream tuples with sequentially increasing identifiers as shown in figure 10.

First a failure is injected on input stream 1 (Failure 1) and then on input stream 3 (Failure 2). Figure 11(a) shows the output when two failures overlap in time and figure 11(b) shows the output when Failure 2 occurs exactly at the moment when Failure 1 heals and node starts reconciling its state.

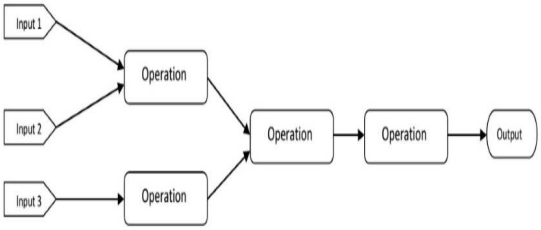

Figure 12: The Query Diagram used in Simultaneous Failures Experiments.

Here the node temporarily loses one or two of its input streams and there are no other replicas that could be reconnected to. When another replica exists for an upstream neighbor the node tries to switch to that replica. When a node switches to a different replica of an upstream neighbor there is a short gap in data it receives. In this prototype implementation we measured that it takes a node approximately 30 milliseconds to switch between upstream neighbors once the node detects failure. Failure detection time depends on the frequency with which downstream node sends keep-alive requests to its upstream neighbors. With a keep-alive period of 100 milliseconds it takes at most 130 milliseconds between the time a failure occurs and the time a downstream node receives data from a different replica of its upstream neighbor. For many application domains it is expected that this value is much smaller than minimum incremental processing latency that the application tolerates. If the application cannot tolerate a short delay the effect of switching upstream neighbors is same as the effect of failure 1 as shown in figure 11(a) but without subsequent failure 2 i.e. the downstream node first suspends and then produces tentative tuples. Once reconnected to new upstream neighbor the downstream node goes back and corrects tentative tuples it produced during the switch. The

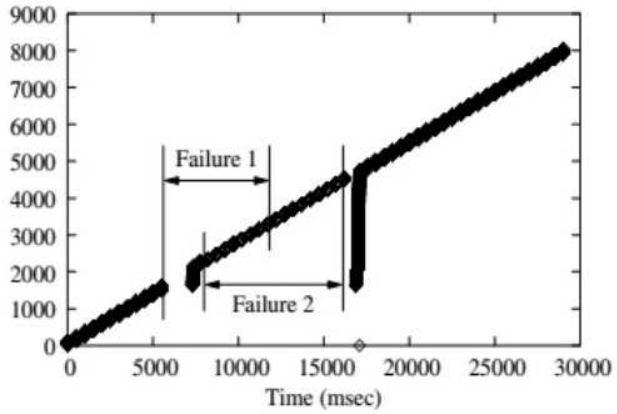

Figure 13: Overlapping Failures.

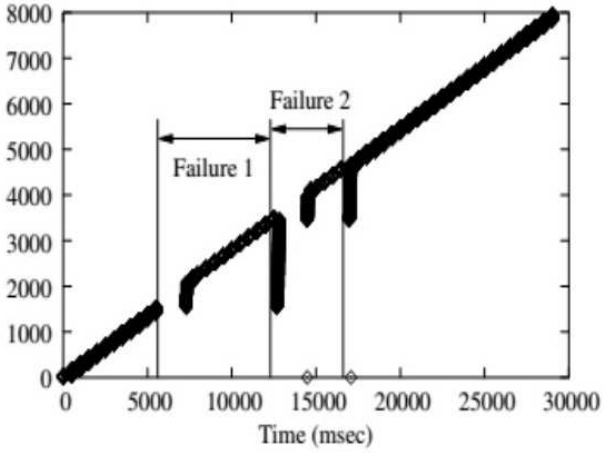

Figure 14: Failure during Recovery

Figure 15: The Outputs with Simultaneous Failures.

maximum incremental latency is set as 3 seconds. Figure 11 shows the maximum gap between new tuples remains below that bound at any time. However node manages to maintain the required availability with sufficiently short reconciliation time.

For longer duration failures reconciliation easily takes longer than the maximum latency bound. The DPC relies on replication to enable a distributed PFRSVM to maintain a low processing latency by ensuring that at least one replica node always processes most recent input data within required bound. To demonstrate this we use the experimental setup as shown in figure 12. We create a failure by temporarily disconnecting one of the input streams without stopping data source. After the failure heals data source replays all missing tuples while continuing to produce new tuples.

The Table 9 in Appendix B shows the maximum processing latency measured at client for failures with

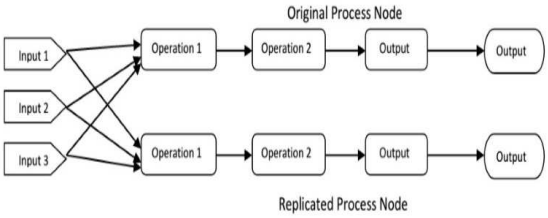

Figure 16: The Experimental Setup for Consistency and Availability tradeoffs for Single Node.

different durations. As the experiment is of deterministic nature each result is an average of three experiments. All values measured are within a few percent of the reported average. The client always receives new data within required 4 second bound. Each node processes input tuples as they arrive without trying to delay them to reduce inconsistency. When the failure first occurs both node replicas suspend for the maximum incremental processing bound and then return to processing tentative tuples as they arrive. After the failure heals DPC ensures that only one replica node at a time reconciles its state while remaining replica nodes continue processing the most recent input data. Once the first replica reconciles its state and catches up with current execution then other replica node reconcile its state in turn. The client application has thus access to the most recent data at all times. The major overhead of DPC lies in buffering tuples during failures in order to replay them during state reconciliation. The overhead is in terms of memory and it does not affect runtime performance. There are however additional sources of overhead that affect runtime performance [24]. To evaluate overhead of these delays the experimental setup is shown in figure 13 which produces appreciable results in terms of failure recovery.

5. Atomicity of transaction: All the data that is generated here through the parallel processing stream archive is automatically consistent with respect to unit of work into the underlying base stream. It is critical that same atomicity property is also supported for archives of serial processing streams in order for recovery and high availability to work correctly. One simple approach to facilitate atomicity is to delay processing of any data until it has been committed. Waiting for commits unfortunately introduces latency within the system. What is really required is the ability to offer a strong guarantee about atomicity and durability of data loaded in the system within a single unit of work without compromising on immediate processing of data. This calls for speculatively processing dirty uncommitted data in a laissez-faire fashion based on the assumption that errors and transaction aborts are few and far between [24]. When a transaction is ac-

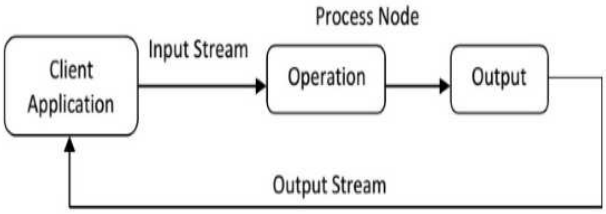

Figure 17: Fault Tolerance Setup.

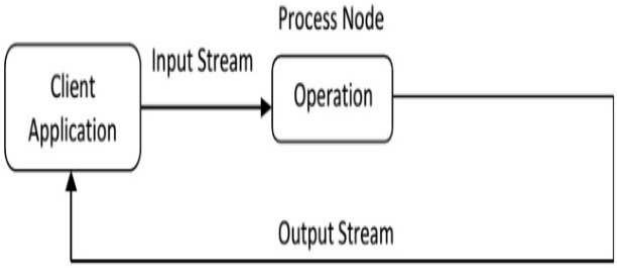

Figure 18: Setup without Fault Tolerance

Figure 19: The Experimental Setup for Fault Tolerance Overhead Experiments.

tually aborted the system asynchronously repairs associated serial processing archives similar to the way repair guarantees order-independence on an eventual consistency basis.

6. Fault tolerance and high availability: The data driven parallel approach used here provides a natural path towards scalability. The next concern is towards enhancing it in order to provide fault tolerance and high availability in the system [32]. The fault tolerance is defined as the ability of system to react well from any kind of extreme or catastrophic error which may happen either from streaming engine itself in the application or in some aspect of hardware or software environment. In particular quick recovery from failure state is critical in realizing fault tolerance. The high availability is characterized as the ability of system to remain up even in the face of any catastrophic error. It is generally realized using additional backup resources that are organized together in either an active-standby or active-active configuration. The unit of work and recovery functionality of the system highlighted earlier serve as key building blocks for fault tolerance and high availability in PFRSVM classification system. This implementation supports a comprehensive fault tolerance and high availability solution by organizing a cluster of FRSVM nodes in the cloud environment in a multi-master active-active configuration. Here same workload are typically running on all nodes of the cluster. Any incoming run of data can be sent to any but only one node in the cluster. It is then the responsibility of a special stream replicator component in each computing node to communicate the complete contents of each run to the peers in the cluster. The runs of data that are populated in a stream by peer are treated just like any other incoming data except that they are not further re-replicated to other nodes. This model has one eventual consistency such that run replication procedure happens on an asynchronous basis. In this way there is very small amount of risk of data loss in the event of any catastrophic media failure between a run getting committed and replicated to peer. The order independent infrastructure plays significantly in realizing simple fault tolerant and high availability architecture. Since each individual node in computing cluster accepts data in any order different nodes stay loosely coupled and implement simple and easy to verify protocols. When a node recovers from failure it immediately starts accepting new transactions and patch up the data it has missed asynchronously. As a node fails in the cluster application layer takes the responsibility in directing all workloads to other nodes. After the failed node is brought back online it captures the entire data it missed while being non-functional. This is accomplished by replicator component using a simple protocol that tracks both replicated and non-replicated runs. PFRSVM executes by parallelizing in-stream dataflow across cluster of workstations in a cost effective way by scaling the high throughput applications. We can imagine having a large number of simultaneous sessions and sources. To keep up with high throughput input rates and low latencies dataflow can be scaled by partitioning it across a cluster. On the cluster in-stream dataflow is a collection of dataflow segments which may be one or more per machine. The individual operations are parallelized by partitioning input and processing across the cluster. When the partitions of an operation need to communicate to non-local partitions of the next operation in the chain communication occurs through exchange architecture [24] as shown in figure 14.

In this configuration a scheme that naively applies dataflow pair technique without accounting for cross machine communication within parallel dataflow quickly becomes unreliable. It is shown that a cluster pair approach leads to a mean-time-to-failure (MTTF) that falls off quadratically with number of machines. Also the parallel dataflow must stall during recovery thereby reducing the availability of system. By embedding the coordination and recovery logic within exchange architecture speeds up mean-time-to-recovery (MTTR) thereby improving both availability and MTTF. The improved MTTF falls off linearly with the number of machines. The flux design achieves the desired MTTF [24] as shown in figure

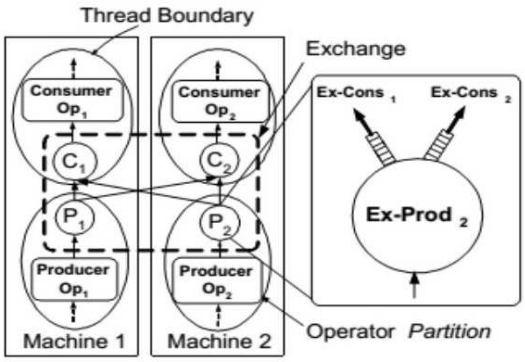

Figure 20: Exchange Architecture.

15.

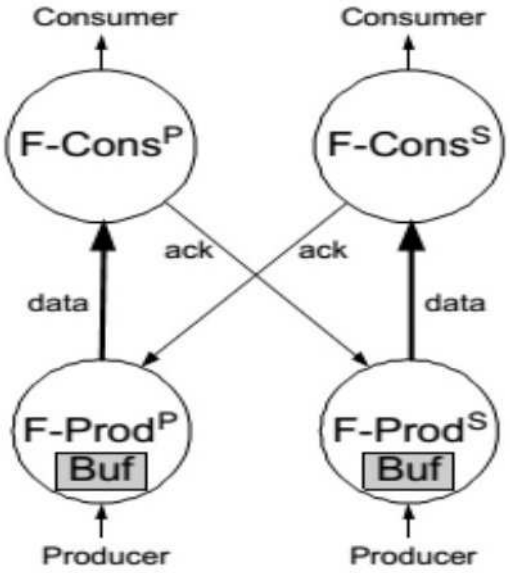

Figure 21: The Flux Design and Normal Case Protocol.

Now we illustrate benefits accrued from the design of PFRSVM by examining performance of parallel implementation of dataflow which highlights its fault tolerant behavior [24]. We implemented streaming group-by aggregation operations, boundary operations and flux within in-stream open source code base. We partition this dataflow across five machines in a cluster and place the operators to perform operations on a separate sixth machine. The operators are supplemented by a duplicate elimination buffer. The flux is inserted after the first group-by operator to repartition its output. Initially we place a partition of each operator on each of the five machines 0 to 4 and replicate them using a chained declustering strategy. Thus each primary partition has its replica on the next machine and last partition has its replica on the first. In this configuration when a single machine fails all five survival scenarios occur in different partitions. At the beginning we introduce a standby machine with operators in their initial state. We did not implemented the controller as it has been implemented by standard cluster management software [24]. We simulate failure by killing the in-stream process on one of those machines which causes connections to that machine to close and raise an error. Each machine is connected to a 100 mbps switch. To approximate workload of high throughput network monitoring the operator generates sequentially numbered session start and end events as fast as possible. With this setup figure 16 shows the total output rate (throughput) and average latency per tuple.

Here the main bottleneck is the network. At t = 20 sec when steady state is reached one of the five machines is killed. The throughput remains steady for some time and then suddenly drops. The drop occurs because during state movement the partition being recovered is stalled and eventually causes all downstream partitions to also stall. Here about 8.9 MB of state was transferred in 941 msec. Once catch up is finished at t = 21 seconds a sudden spike in throughput is observed. This spike occurs because during movement all queues to the unaffected partitions are filled and ready to be processed once catch-up completes. Figure 16 shows an increase in latency because input and in-fight data are buffered during movement. Then the output rate and average latency settle down to normal. During this entire process the input rate stayed at constant 42000 tuples/sec with no data dropped.

This investigation illustrates that with piecemeal recovery and sufficient buffering we can effectively mask the effects of machine failures. To understand the overheads of flux we added enough CPU processing to the lower level group-by to make the bottleneck. Here for single parallel dataflow the input rate was 83000 tuples/sec; for cluster pairs it was 42000 tuples/sec and for flux it was 37000 tuples/sec. Additional processing only reduces flux overhead relative to others.

7. Bootstrapping the system: When a live workload is added to the streaming classification system on an ad-hoc basis it is easy to see only the data that arrives after the request is submitted. This approach is quite reasonable if the request involves small windows involving few seconds only. In many situations this is not the case as requests may continue for hours and a naive streaming implementation does not produce complete results until steady state is reached. This entails bootstrapping feature requirement from the system [24]. This exploits any available archives of the underly-

ing raw or derived stream that workload is based on by reaching into the archive and replaying history in order to build up runtime state of the request and produce complete results as early as possible. In addition, if a new workload has an associated archive there is often a need for the system to populate this archive with all data already in the system prior to the request.

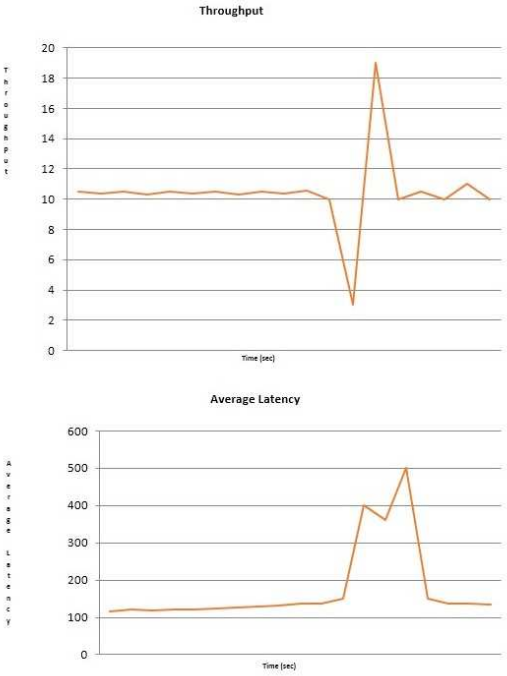

Figure 22: System Performance (Throughput and Average Latency per Tuple).

9 Conclusion

In this work we propose a robust parallel version of fuzzy support vector machine i.e. PFRSVM to classify and analyze useful information from huge datasets present in cloud environment. It is an in-stream data classification engine which adheres to the fundamental rules of stream processing. The classifier is sensitive to noisy data samples. The fuzzy rough membership function brings sensitivity to noisy samples and handles impreciseness in training samples giving robust results. The decision surface sampling is achieved through membership function. The larger membership function value increases the importance of corresponding point. The classification success lies in proper selection of fuzzy rough membership function. PFRSVM success is attributed towards choosing appropriate parameter values. The training samples are either linear or nonlinear separable. In nonlinear training samples, using linear separating input space is mapped into high dimensional feature space to compute separating surface. PFRSVM effectively addresses nonlinear classification and imbalance and overlapping class problems. It generalizes to unseen data and relaxes dependency between features and labels. PFRSVM performance is also assessed in terms of number of support vectors. We use MapReduce technique to improve scalability and parallelism of split dataset training. The research work is performed on cloud environment available at University of Technology and Management, India on real datasets from UCI Machine Learning Repository. The experiments illustrate the convergence of PFRSVM. The performance and generalization of the algorithm are evaluated through Hadoop. The algorithm works on cloud systems dealing with large scale dataset training problems without having any knowledge about the number of computers connected to run in parallel. The experimental results have been reported here for $C = 8$ and 128 using Gaussian RBF and Bayesian kernels. For Gaussian kernel both PFRSVM and FRSVM achieve highest test rate. For Bayesian kernel both algorithms have better generalization performance for Ionosphere and Semeion Handwritten Digit datasets. Further PFRSVM has better generalization ability than FRSVM. PRFSVM also has better performance on reducing the outliers' effects. Empirically prediction variability, generalization performance and risk minimization of PFRSVM is better than existing SVMs. The average classification accuracy of PFRSVM is better than SVM, FSVM and MFSVM for Gaussian RBF and Bayesian kernels. The experimental results on both synthetic and real datasets clearly demonstrate the superiority of proposed technique. The stream classification engine is scalable and reliable and maintains order independence in data processing, streamlines computational transaction, recovers from failure, generates atomic transactions and are fault tolerant and highly available in nature.

Acknowledgement

The author extends his gratitude to the technical staff of Samsung Research and Development Institute Delhi, India for providing support through the entire course of this research work.

References

[1] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh and A. H. Byers, Big Data: The next frontier for innovation, competition, and productivity, Technical Report, McKinsey Global Institute, McKinsey and Company, 2011.

[2] M. B. Miles, M. A. Huberman and J. Saldaña, *Qualitative data analysis*: *A methods sourcebook*. SAGE Publications, 2014.

[3] J. Han, M. Kamber and J. Pei, *Data mining*: *Concepts and techniques*, San Francisco: Morgan Kaufmann Publishers, 2011.

[4] J. Canny and H. Zhao, Big data analytics with small footprint: Squaring the cloud, *In 2013 Proc. Nineteenth ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pp. 95–103.

[5] C. Statchuk and D. Rope, Enhancing Enterprise Systems with Big Data, Technical Report, IBM Business Analytics Group, IBM Corporation, 2013.

[6] J. Leskovec, A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*. Cambridge University Press, 2014.

[7] HDFS (Hadoop Distributed File System) Architecture: http://hadoop.apache.org/common/docs/ current/hdfs_design.html, 2009.

[8] K. Hwang, G. C. Fox and J. J. Dongarra, *Distributed and Cloud Computing*: *From Parallel Processing to Internet of Things*. Morgan Kaufmann, 2011.

[9] E. Capriolo, D. Wampler and J. Rutherglen, *Programming Hive*. O'Reilly Media, 2012.

[10] J. Abonyi, B. Feil and A. Abraham, Computational Intelligence in Data Mining, Informatica, vol. 29, no. 1, pp. 3—12, 2005.

[11] D. Dubois and H. Prade, Putting Rough Sets and Fuzzy Sets together, In R. Slowinski (Editor) *Intelligent Decision Support*, Handbook of Applications and Advances of the Rough Set Theory, pp. 203—232, Kluwer Academic Publishers, 1992.

[12] C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, vol. 2, no. 2, pp. 121—167, 1998.

[13] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer Verlag, 2007.

[14] Q. Hu, S. An, X. Yu and D. Yu, Robust Fuzzy Rough Classifiers, Fuzzy Sets and Systems, vol. 183, no.1, pp. 26—43, 2011.

[15] A. Chaudhuri, *Data Classification through Fuzzy and Rough versions of Support Vector Machines*: *A Survey*. Technical Report, Samsung Research and Development Institute Delhi, 2014.

[16] V.N. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.

[17] A. Chaudhuri, K. De, and D. Chatterjee, A Comparative Study of Kernels for Multi-Class Support Vector Machine, *In 2008 Proc. Fourth Conf. on Natural Computation*, vol. 2, pp. 3—7.

[18] A. Chaudhuri and K. De, Fuzzy Support Vector Machine for Bankruptcy Prediction, Applied Soft Computing, vol. 11, no. 2, pp. 2472—2486, 2011.

[19] A. Chaudhuri, Modified Support Vector Machine for Credit Approval Classification, AI Communications, vol. 27, no. 2, pp. 189—211, 2014.

[20] H. J. Zimmermann, *Fuzzy Set Theory and its Applications*. Boston: Kluwer Academic, 2001.

[21] S. Perera and T. Gunarathne, *Hadoop MapReduce Cookbook*. Packt Publishers, 2013.

[22] R. Bekkerman, M. Bilenko and J. Langford, *Scalable Machine Learning*. Cambridge University Press, 2012.

[23] C. C. Chang and C. J. Lin, LIBSVM: A Library for Support Vector Machines, ACM Transactions on Intelligent Systems and Technology, vol. 2, no. 3, pp. 1—27, 2011.

[24] A. Chaudhuri, *Studies on Parallel SVM based on MapReduce*. Technical Report, Birla Institute of Technology Mesra, Patna Campus, India, 2010.

[25] I. W. Tsang, J. T. Kwok and P. M. Cheung, Core Vector Machines: Fast SVM Training on very Large Datasets, Journal of Machine Learning Research, vol. 6, pp. 363—392, 2005.

[26] S. Ramaswamy, R. Rastogi and K. Shim, Efficient Algorithms for Mining Outliers from Large Datasets, *In 2000 Proc. ACM SIGMOD Conf. on Management of Data*, pp. 427—438.

[27] V. Punyakanok, D. Roth, W. Tau Yih and D. Zimak, Learning and Inference over Constrained Output, *In 2005 Proc. 19th Joint Conf. on Artificial Intelligence*, pp. 1124—1129.

[28] B. Ellis, *Real Time Analytics*: *Techniques to Analyze and Visualize Streaming Data*. John Wiley and Sons, 2014.

[29] M. Stonebraker, U. Cetintemel and S. Zdonik, *The Eight Rules of Real Time Stream Processing*. White Paper, StreamBase Systems, MA, United States, 2010.

[30] J. L. Hennessy and D. A. Patterson, *Computer Architecture – A Quantitative Approach*. 5th Edition, Morgan Kaufmann Publications, Elsevier Inc., 2012.

[31] Borealis: Second Generation Stream Processing Engine: http://nms.lcs.mit.edu/projects/borealis, 2003.

[32] R. Jhawar, V. Piuri and M. Santambrogio, Fault Tolerance Management in Cloud Computing: A System Level Perspective, IEEE Systems Journal, vol. 7, no. 2, pp. 288-–297, 2013.

Appendix A

$rm_A(p) = \frac{\|[p]_R \cap A\|}{\|[p]_R\|}$; $\|\mathrm{A}\|$ is cardinality of set A (4)

$$\mu_{\underline{C_c}}(FC_i) = \underbrace{inf}_{x \in C_c} max\left\{1 - \mu_{FC_i}(p), \mu_{C_c}(p)\right\} \quad \forall p \quad (5)$$

$$\min \quad \frac{1}{2}\|\mathrm{w}\|^2 + C\sum_{i=1}^{M} frm_i(p)\xi_i$$

$$\text{subject to: } \begin{cases} z_i\left(\mathrm{w}^{\mathrm{T}}\Phi(Y_i) + b\right) \geq 1 - \xi_i, i = 1, .., M; \\ \xi_i \geq 0 \end{cases} \quad (10)$$

$$rd_+^2 = \frac{1}{n}\max\left\|\Phi(Y') - \Phi_+\right\|^2 = \frac{1}{n}\max\left[\Phi^2(Y') - 2\Phi(Y')\cdot\Phi_+ + \Phi_+^2\right] =$$

$$\frac{1}{n}\max\begin{bmatrix} \Phi^2(Y') - \frac{2}{m_+}\sum_{Y_i \in C^+} tanh\left[\Phi(Y_i)\cdot\Phi(Y')\right] + \\ \frac{1}{m_+^2}\sum_{Y_i \in C^+}\sum_{Y_j \in C^+} tanh\left[\Phi(Y_i)\cdot\Phi(Y_j)\right] \end{bmatrix}$$

$$= \frac{1}{n}\max\begin{bmatrix} K(Y', Y') - \frac{2}{m_+}\sum_{Y_i \in C^+} K(Y_i, Y') + \\ \frac{1}{m_+^2}\sum_{Y_i \in C^+}\sum_{Y_j \in C^+} K(Y_i, Y_j) \end{bmatrix} \quad (15)$$

$$rd_-^2 = \frac{1}{n}\max\left[K(Y', Y') - \frac{2}{m_-}\sum_{Y_i \in C^-} K(Y_i, Y') + \frac{1}{m_-^2}\sum_{Y_i \in C^-}\sum_{Y_j \in C^-} K(Y_i, Y_j)\right] \quad (16)$$

$$frm_i(p) = \begin{cases} 1 - \left(\frac{\sum_{i=1}^{H}\mu_{FC_i}(p)\tau_{C_c}^i}{\sum_i \mu_{FC_i}(p)}\right)\sqrt{\frac{\left\|dist_{i+}^2\right\| - \left\|dist_{i+}^2\right\|\cdot rd_+^2 + rd_+^2}{\left(\left\|dist_{i+}^2\right\| + \left\|dist_{i+}^2\right\|\cdot rd_+^2 + rd_+^2\right) + \varepsilon}} & (z_i = 1) \\ 1 - \left(\frac{\sum_{i=1}^{H}\mu_{FC_i}(p)\tau_{C_c}^i}{\sum_i \mu_{FC_i}(p)}\right)\sqrt{\frac{\left\|dist_{i-}^2\right\| - \left\|dist_{i-}^2\right\|\cdot rd_-^2 + rd_-^2}{\left(\left\|dist_{i-}^2\right\| + \left\|dist_{i-}^2\right\|\cdot rd_-^2 + rd_-^2\right) + \varepsilon}} & (z_i = -1) \end{cases} \quad (19)$$

$$frm_i^c(p) = \begin{cases} 1 - \left(\frac{\sum_{i=1}^{H}\mu_{FC_i}(p)\tau_{C_c}^i}{H}\right)\sqrt{\frac{\left\|dist_{i+}^2\right\| - \left\|dist_{i+}^2\right\|\cdot rd_+^2 + rd_+^2}{\left(\left\|dist_{i+}^2\right\| + \left\|dist_{i+}^2\right\|\cdot rd_+^2 + rd_+^2\right) + \varepsilon}} & (z_i = 1) \\ 0 & (z_i = -1) \end{cases} \quad (20)$$

$$\max H^{i,j} = -\frac{1}{2}\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix}^T \begin{bmatrix} KM_{11} & KM_{12} \\ KM_{21} & KM_{22} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}^T \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix}$$

subject to:

$$0 \leq \alpha_i \leq C \ \ \forall i \ \ \wedge \ \ \sum_{i=1}^{I} \alpha_i z_i = 0 \quad (26)$$

$$R(\theta) = \{X : |(FR(\omega_1|X) - FR(\omega_2|X))| < \theta\} \quad (27)$$

Appendix B

Dataset	Characteristics	No. of Instances	No. of Attributes
German Credit	Multivariate, Categorical, Integer	1000	20
Heart Disease	Multivariate, Categorical, Integer	303	75
Ionosphere	Multivariate, Integer	351	34
Semeion Handwritten Digit	Multivariate, Integer	1593	256
Landsat Satellite	Multivariate, Integer	6435	36

Table 2: The UCI Machine Learning Datasets used.

	SVM	PFRSVM	0.5 % samples
Number of data points	114996	572	602
SVM training time (sec)	160.796	0.002	0.002
Sampling time (sec)	0.0	9.569	3.114
Number of false predictions (Number of false positives, Number of false negatives)	75 (50, 25)	69 (58, 11)	237 (219, 18)

Table 3: Experimental Results on Synthetic Dataset (Number of Training Data = 124996, Number of Testing Data = 107096).

Sampling rate (%)	Number of data	Number of errors	Training time	Sampling time
0.0001	24	6423	0.000114	823.00
0.001	230	2399	0.000969	825.00
0.01	2344	1125	0.02	828.00
0.1	23475	1009	6.286	834.25
1	234386	1014	1189.786	838.86
5	1151719	1019	20705.4	842.69
MFSVM	2308	859	2.969	2796.214
PFRSVM	3896	862	1.422	2236.219

Table 4: Experimental Results on Large Synthetic Dataset (Number of Training Data = 24069196, Number of Testing Data = 242896).

PFRSVM with Gaussian RBF Kernel			
Datasets	Support Vectors	Training Rate %	Test Rate %
German Credit	606	85.0	86.0
Heart Disease	137	65.5	73.7
Ionosphere	160	69.0	77.7
Semeion Handwritten Digit	737	89.4	93.5
Landsat Satellite	1384	95.0	97.0
FRSVM with Gaussian RBF Kernel			
German Credit	636	79.7	78.9
Heart Disease	165	60.7	69.0
Ionosphere	180	65.0	74.9
Semeion Handwritten Digit	765	84.5	86.8
Landsat Satellite	1407	93.0	93.7
PFRSVM with Bayesian Kernel			
German Credit	640	89.0	85.7
Heart Disease	145	67.7	69.9
Ionosphere	157	70.7	75.0
Semeion Handwritten Digit	735	89.0	89.7
Landsat Satellite	1396	96.9	96.9
FRSVM with Bayesian Kernel			
German Credit	645	83.0	78.0
Heart Disease	175	64.8	65.8
Ionosphere	177	67.0	70.7
Semeion Handwritten Digit	755	86.8	83.8
Landsat Satellite	1425	96.0	89.8

Table 5: Experimental Results of PFRSVM and FRSVM on different Datasets with $C = 8$.

PFRSVM with Gaussian RBF Kernel			
Datasets	Support Vectors	Training Rate %	Test Rate %
German Credit	596	95.2	96.0
Heart Disease	127	77.5	88.7
Ionosphere	142	82.0	92.7
Semeion Handwritten Digit	727	97.9	94.5
Landsat Satellite	1362	96.0	97.2
FRSVM with Gaussian RBF Kernel			
German Credit	625	94.0	93.5
Heart Disease	154	75.0	84.7
Ionosphere	172	80.9	89.7
Semeion Handwritten Digit	754	97.7	91.5
Landsat Satellite	1392	94.2	94.6
PFRSVM with Bayesian Kernel			
German Credit	632	97.2	95.7
Heart Disease	147	82.7	82.8
Ionosphere	145	86.7	87.2
Semeion Handwritten Digit	725	98.0	98.6
Landsat Satellite	1386	97.0	97.6
FRSVM with Bayesian Kernel			
German Credit	635	96.9	93.7
Heart Disease	165	79.9	80.0
Ionosphere	165	83.0	85.0
Semeion Handwritten Digit	745	97.7	97.0
Landsat Satellite	1402	95.2	95.7

Table 6: Experimental Results of PFRSVM and FRSVM on different Datasets with $C = 128$.

Dataset	SVM	FSVM	MFSVM	PFRSVM
German Credit	78.3	80.5	82.9	94.8
Heart Disease	83.5	86.9	87.0	95.9
Ionosphere	85.7	87.7	89.0	96.5
Semeion Handwritten Digit	82.7	86.0	86.5	95.5
Landsat Satellite	86.8	86.9	87.2	96.2

Table 7: The Average Classification Accuracy Percentage of different versions of SVM with Gaussian RBF Kernel.

Dataset	SVM	FSVM	MFSVM	PFRSVM
German Credit	77.7	78.0	80.7	94.2
Heart Disease	83.4	85.5	86.0	94.9
Ionosphere	85.5	86.0	87.9	95.5
Semeion Handwritten Digit	80.8	85.0	85.8	94.7
Landsat Satellite	86.9	87.2	87.8	96.6

Table 8: The Average Classification Accuracy Percentage of different Versions of SVM with Bayesian Kernel.

Failure Duration (seconds)	Maximum Processing Latency (seconds)
2	2.1
4	2.7
6	2.7
8	2.7
10	2.7
12	2.7
14	2.7
16	2.7
30	2.7
45	2.7
60	2.7

Table 9: The Maximum Processing Latency for different Failure Durations and for Single Node Deployment with One Replica.