Load Shedding in Classifying Multi-Source Streaming Data: A Bayes Risk Approach

Yijian Bai
UCLA
bai@cs.ucla.edu

Haixun Wang
IBM T. J. Watson
haixun@us.ibm.com

Carlo Zaniolo
UCLA
zaniolo@cs.ucla.edu

Abstract

Monitoring multiple streaming sources for collective decision making presents several challenges. First, streaming data are often of large volume, fast speed, and highly bursty nature. Second, it is impossible to offload classification decisions to individual data sources, each of which lacks full knowledge for the decision making. Hence, the central classifier responsible for decision making may be frequently overloaded. In this paper, we study intelligent load shedding for classifying multi-source data. We aim at maximizing classification quality under resource (CPU and bandwidth) constraints. We use a Markov model to predict the distribution of feature values over time. Then, leveraging Bayesian decision theory, we use Bayes risk analysis to model the variances among different data sources in their contributions to the classification quality. We adopt an Expected Observational Risk criterion to quantify the loss of classification quality due to load shedding, and propose a Best Feature First (BFF) algorithm that greedily minimizes such risk. The effectiveness of the approach proposed is confirmed by experiments.

1 Introduction

Mining high-speed, large volume data streams introduces new challenges for resource management [6, 9]. In many applications, data from multiple sources arrive continuously at a central processing site, which analyzes the data for knowledge-based decision making. Under overloaded situations, policies of *load shedding* must be developed for incoming data streams so that the quality of decision making is least affected.

Multi-task, Multi-source Stream Classification Consider a central sever that handles n independent classification tasks, where each task has multiple input streams (Figure 1)[1]. Our problem is the following. *Suppose that, at a given moment, the central classifier, which monitors $n \times k$ streams from n tasks, only has capacity to process m out of the $n \times k$ input streams. Then, which input streams should we pick to maximize the classification quality?* We use the following two examples to illustrate situations that give rise to the problem.

- A security application monitors many locations. At each location, multiple cameras monitor from different viewing angles to precisely determine the speed and direction of moving objects. In this case, data from different cameras are of the same type but they have different semantics (different viewing angles).
- In environment monitoring, a central classifier makes decisions based on a set of factors, such as temperature, humidity, etc., each obtained by sensors distributed in a wireless network. In this case, multiple data sources for one task contains different types of information.

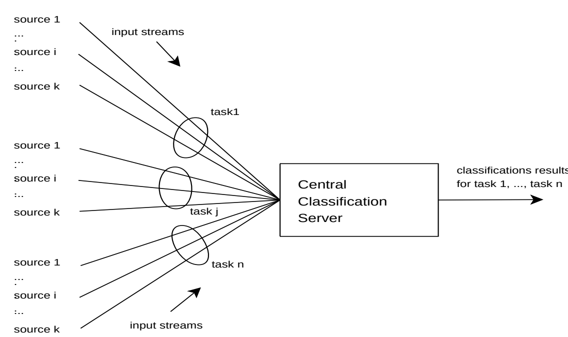


Figure 1: Multi-task, Multi-source Stream Classification

An inherent challenge to this problem is that the decision making cannot be easily offloaded to each data source, as classification depends on information from all of the k sources. On the other hand, in most situations, at any given time there exist only a small number of events of potential interest (e.g., a small number of monitored locations have abnormal activities). This means that, even if $m \ll n \times k$, it is still possible to monitor all the tasks and catch all events of interest, if we know how to intelligently shed loads.

The following factors may have significant implications on the overall cost of the classification process. i) *Cost of data preprocessing.* Raw data from the sources may have to be preprocessed before classification algorithms can be applied. (E.g., extracting objects from video frames, which can be a very costly process.) ii) *Cost of data transmission.* Delivering large amount of data from remote sources to the centralized server may incur considerable cost. iii) *Cost of data collection.* Data may be costly to obtain to begin with.

[1] For presentation simplicity, we assume that every task has k input streams and each stream provides data on one *feature* used for classification. Although, each input stream may in fact contain one or more *features* in reality, to which case our method can be easily generalized.

This may limit the sampling rate of a sensor, or its on-line time due to energy conservation concerns.

As a concrete example, the central server in the above security application may have to perform a two-step procedure: a) the server *observes* the video stream, i.e., receives the stream from the network, parses video frame images to determine the composition and location of objects, which has a very high computation cost in transmission and parsing. b) the server runs a classification algorithm on the interpreted image to determine potential security risk.

When any of the above data acquiring/observation costs are the dominant factors in the process, it becomes worthwhile to pay a reasonable cost to optimize load-shedding.

State-of-the-Art Approaches None of the existing solutions fully address challenges associated with our problem.

- *Randomly shedding load*
  Dropping data indiscriminately and randomly [7, 2, 1] may lead to degradation of classification quality, as not all incoming data contribute equally to the quality.
- *Solving the special case of* $k = 1$
  LoadStar [4] assumes that each classification task has only one data source ($k = 1$). The load shedding decisions are made on a task-by-task basis, and it does not consider that different features of the same task may contribute differently to the overall quality. In fact for $k = 1$, we can safely offload load shedding decisions to data sources, which already have complete information.

Observations We introduce a quality-aware load shedding mechanism based on the following observations.

1. Streaming data often exhibit strong temporal-locality (e.g., videos showing objects' movement over time). This property enables us to build a model to predict the content of the next incoming data.
2. At a given time, multiple sources (features) of a task may have different degrees of impact on the classification result. For example, an approaching object may be caught by a camera at one angle much earlier than other angles. Therefore, we should choose to observe features contributing the most to accuracy.
3. In the Bayesian decision theory, *Bayes Risk* is used to measure the quality of classification, and prevent errors that are most costly for particular applications [5] (e.g., false alarms in a security application is usually more acceptable than missing alarms). We argue that the load-shedder for the classifier must try to prevent the same type of errors. We show that the optimal guideline for load-shedding is limited to a part of the Bayes Risk that is caused solely by the lack of data observation, which we term the Observational Risk.

Contributions. To the best of our knowledge, this is the first report that studies load shedding for the general multi-source stream classification problem. Our paper makes the following contributions. (a) We propose *feature-based* load shedding using Observational Risk as the guideline. (b) We give a complete analysis of the Bayes risk and a novel algorithm BFF that greedily minimizes the expected Observational Risk on a feature-by-feature basis. (c) We present experiments data to show the effectiveness of our algorithms.

2 Problem Analysis and the Markov Model

Consider classification tasks that monitor two data sources (i.e., two features) X_1 and X_2; therefore, their states can be modeled as points in a two-dimensional feature space. In Figure 2, we show three such tasks at time t, namely, $A(t)$, $B(t)$, and $C(t)$. The feature space is divided into two classes — inside and outside the shaded area.

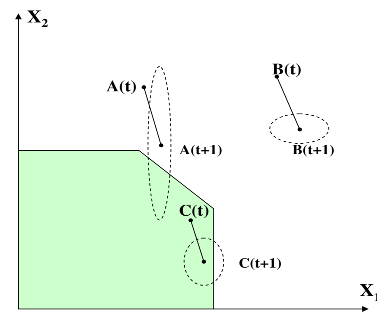


Figure 2: Task Movement in the Feature Space

Let p be the probability distribution of a point's position at time $t + 1$. Assuming that the two features X_1 and X_2 are independently distributed as normal distributions, the position of a point at time $t + 1$ is within an elliptical boundary with high probability. Then we can draw the following conclusions based on p, which will be formalized in Section 3.

1. Different tasks should be given different priorities. For example, according to p, no matter where B moves to, its classification result stay the same with high confidence, thus we can safely predict its class label without any observation. This is not true for A and C.
2. Different features (streams) should be given different priorities. Intuitively, for task A, observations of feature X_2 is more critical, and for task C, observations of feature X_1 is more critical. In Figure 3(a), we zoom in on task A. Suppose we can only afford to observe one feature out of the two. If we observe X_2 and get expected value x_2, then the distribution p degenerates into a horizontal line segment in Figure 3(c), representing the conditional distribution $p(X_1|X_2 = x_2)$. The resulted distribution does not run across the decision boundary — i.e., with high confidence no matter what the value of X_2 happens to be, the classification will be the same. This allows us to make a prediction without observing X_1. However, if we instead choose X_1 and get expected value x_1, then the curve for $p(X_2|X_1 = x_1)$, shown in Figure 3(b), still runs across the decision boundary, and we are unable to classify A with high confidence.

Markov Model for Movement in Feature Space Assuming that a point's location in the feature space at time $t+1$ is solely dependent on its location at time t, we use a finite

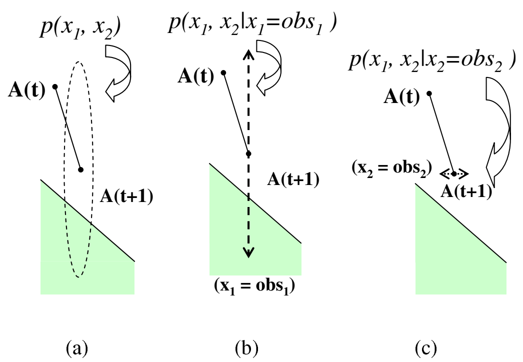


Figure 3: Joint and Conditional Distributions

discrete-time Markov chain to model a point's movement as a stochastic process, in order to learn the distribution p at time $t+1$. We also assume that features are independent with respect to movement, thus we build a Markov model on each feature (instead of a multivariate Markov model, which may require a very large transition matrix). More specifically, let X be a feature that has *M* distinct values (continuous values are discretized), our goal is to learn a state transition matrix K of size $M \times M$, where entry K_{ij} is the probability that feature X will take value j at time $t+1$ given $X = i$ at time t.

The MLE (maximum likelihood estimation) of the transition matrix K is given by:

$$\hat{K}_{ij} = \frac{n_{ij}}{\sum_k n_{ik}}$$

I.e., the fraction of observed transitions from i to j among transitions from i to k, for all possible k. We use a finite sliding window of recent history for this estimation to accommodate concept drifts in streaming data.

3 Bayes Risk Analysis

In this section, we argue that a portion of the expected Bayes Risk, which we call the expected *Observational Risk*, should be used as the metric for feature-based load shedding.

3.1 The Expected Bayes Risk Let $\delta(c_i|c_j)$ denote the cost of predicting class c_i when the data is really of class c_j. Then, at a given point $\vec{x}$ in the feature space, the risk of our decision to label $\vec{x}$ as class c_i out of K classes is:

$$R(c_i|\vec{x}) = \sum_{j=1}^{K} \delta(c_i|c_j)P(c_j|\vec{x}) \tag{3.1}$$

where $P(c_j|\vec{x})$ is the posterior probability that $\vec{x}$ belongs to class c_j. One particular loss function is the zero-one loss function, which is given by

$$\delta(c_i|c_j) = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{if } i \neq j \end{cases}$$

under which, the conditional risk in Eq 3.1 becomes

$$R(c_i|\vec{x}) = 1 - P(c_i|\vec{x}) \tag{3.2}$$

Risk Before Feature Observation Without any observation, our knowledge about a point's next location in the feature space is completely specified by distribution $p(\vec{x})$. Therefore the expected risk for classifying a point $\vec{x}$ as class c_i is:

$$R_{before}(c_i) = E_{\vec{x}}[R(c_i|\vec{x})] = \int_{\vec{x}} R(c_i|\vec{x})p(\vec{x})d\vec{x} \tag{3.3}$$

which computes an integration over the elliptical area in Figure 3(a). Note $p(\vec{x})$ is a shorthand for $p_{t+1}(\vec{x})$, which is derived from the current distribution and the state transition matrix K of the Markov model as: $p_{t+1}(\vec{x}) = p_t(\vec{x})K$. The best prediction c_k thus minimizes the expected risk:

$$k = \underset{i}{\operatorname{argmin}}\, R_{before}(c_i) = \underset{i}{\operatorname{argmin}}\, E_{\vec{x}}[R(c_i|\vec{x})] \tag{3.4}$$

Therefore, the expected risk before any observation is $R_{before}(c_k)$, i.e., the risk associated with the best prediction.

Risk After Feature Observation Suppose we observe $x_j = obs_j$, the total risk for labeling this partially observed data point as class c_i becomes[2]:

$$\begin{aligned} R_{after}(c_i|obs_j) &= E_{(\vec{x}|x_j=obs_j)}[R(c_i|\vec{x})] \\ &= \int_{\vec{x}|x_j=obs_j} R(c_i|\vec{x})p(\vec{x}|obs_j)d\vec{x} \end{aligned} \tag{3.5}$$

Clearly, Figure 3(b) and 3(c) correspond to Eq 3.5 with different x_j observations, where the resulted risks are integrated over different areas.

Risk Reduction due to Observation The benefit of making an observation of x_j is given by the reduction in the expected Bayes Risk. Suppose after observation the class that minimizes the risk is c'_k, then the expected risk is $R_{after}(c'_k|obs_j)$, and we have

$$R_{diff}(obs_j) = R_{before}(c_k) - R_{after}(c'_k|obs_j) \tag{3.6}$$

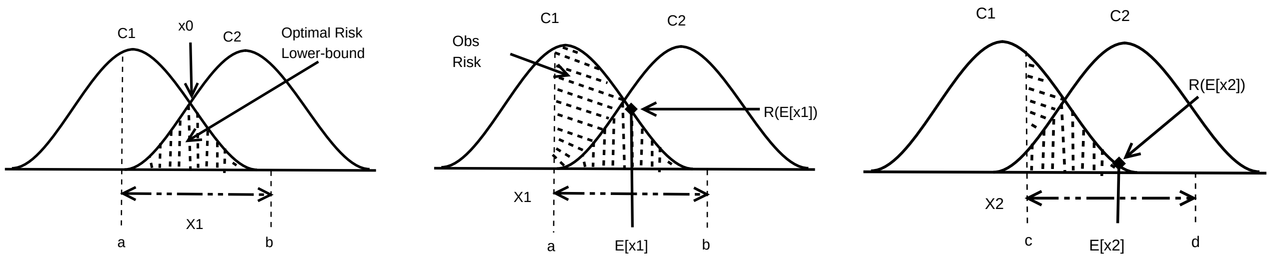
Thus, a greedy method would choose the feature that maximizes Eq 3.6 among all features from all tasks[3]. Therefore, the best feature is:

$$j^* = \underset{j}{\operatorname{argmax}}\, R_{diff}(obs_j) \tag{3.7}$$

Quality of Feature Observation Eq 3.6 provides a guideline for feature observation in load shedding. However, the observed value obs_j is unknown at the time of load-shedding, thus we substitute obs_j by the expected value of the feature, $E[x_j]$, as our best guess for the observation. This leads to the following Quality of Observation (QoO) metric

[2]Sometimes we use $p(\vec{x}|obs_j)$ to stand for $p(\vec{x}|X_j = obs_j)$ for ease of presentation.

[3]Note that, the predicted classes before any observation, c_k, is task-dependent. I.e., the $R_{before}(c_k)$ should really be $R_{before}(c_k; task_{obs_j})$, where observation obs_j belongs to $task_{obs_j}$. Therefore c_k is shared by all obs_j for the same task, but different in different tasks. Same applies to equation 3.8.



(a) The Optimal Risk Lower-bound of Feature X1 for time $t+1$

(b) Risk Decomposition and Expected Risk of Feature X1 for Time t

(c) Risk Decomposition and Expected Risk of Feature X2 for Time t

Figure 4: Bayes Risk Decomposition

definition, where the quality of making an observation on feature X_j is conditioned upon its expected value, as follows:

$$Q_{Bayes}(X_j) = R_{before}(c_k) - R_{after}(c'_k|E[x_j]) \tag{3.8}$$

A generalized metric for making the k-th feature observation after already having observed $k-1$ features can be derived in a similar manner.

3.2 The Expected Observational Risk There is a pitfall in directly using the expected Bayes Risk for load shedding, as shown next.

Dissecting the risk Let $p(C_1|x)$ and $p(C_2|x)$ be the posterior distributions of two classes C_1 and C_2. Without loss of generality, Figure 4(a) shows the two distributions as two bell curves. At point x_0, we have $p(C_1|x) = p(C_2|x)$. Therefore, x_0 is the classification boundary of C_1 and C_2.

At time $t+1$, if $X_1 = x_1$, assuming 0/1 loss, the optimal risk at x_1 is the value of the smaller posterior probability. (For graph illustration purpose, assume here that, the feature values of X_1 at time $t+1$ has a uniform distribution within range $[a, b]$.) Therefore, the expected optimal risk is the average of the shaded area in Figure 4(a). This expected optimal risk cannot be further reduced by improving the underlying classifier, or by any other means. In fact, it is the unavoidable, lowest risk, as it is dictated by the overlapping nature of the class posterior probabilities.

Then, suppose we need to predict the value of X_1 at time $t+1$. Although C_1 should have been the optimal class, if we predict X_1 to be any value less than x_0, we will instead classify the task as C_2. Then the total Bayes Risk is the shaded areas in Figure 4(b). Compared with the optimal risk, the increased portion, which we call the Observational Risk, is shown as the extra shaded area, which is solely brought by wrong predictions on X_1. By observing the value of a feature, we can eliminate the Observational Risk associated with that feature.

The Pitfall We compare X_1 in Figure 4(b) and X_2 in Figure 4(c). At time $t+1$, X_2 has a distribution that is uniform within $[c, d]$, and is different from that of X_1. As shown in the figure, we should choose to observe feature X_1, since its area of the Observational Risk is larger. However, if we use Eq 3.8 in the last section, since $R(E[X_1])$ is much larger than $R(E[X_2])$, the Bayes Risk reduction likely will favor X_2.

The Q_{Obs} Metric: Therefore, we modify our Quality of Observation metric Q_{Bayes} into Q_{Obs}, which only measures the reduction of the expected *Observational Risk* by data observation. The general metric of Q_{Obs}, shown next, measures the quality of the k_{th} observation x_k, after having already observed a total of $k-1$ features. This metric is conditioned on the feature values we have already observed so far ($obs_1, obs_2, \cdots, obs_{k-1}$), and the expected value of the feature x_k that we are about to observe, as follows:

$$\begin{aligned} Q_{Obs}(X_k) &= \int_{\vec{x}|obs_{1,\cdots,k-1}} R'(c_i|\vec{x})p(\vec{x}|obs_{1,\cdots,k-1})d\vec{x} \\ &- \int_{\substack{\vec{x}|obs_{1,\cdots,k-1}, \\ x_k=E[x_k]}} R'(c'_i|\vec{x})p(\vec{x}|obs_{1,\cdots,k-1}, E[x_k])d\vec{x} \end{aligned} \tag{3.9}$$

Here $R'(c_i|\vec{x})$ stands for $[R(c_i|\vec{x}) - R(c^*|\vec{x})]$: where c_i is the best predicted class based on currently-known data distribution (by Eq 3.4); and c^* is the optimal class label at the particular location $\vec{x}$, obtained based on the class posterior distributions. This metric Q_{Obs} strictly optimizes the portion of the Bayes Risk that is reducible by observation. The analytic derivation of this metric can be found at [3] and is omitted here.

4 The Best Feature First (BFF) Algorithm

The Best Feature First (BFF) algorithm (shown in Algorithm 1) is derived based on Eq 3.9. At the beginning of each time unit we first compute the predicted distributions for each feature using Markov chains, and then repeatedly pick to observe the *best* unobserved feature over all tasks that leads to the largest reduction in expected *Observational Risk*. By doing so, we greedily minimize the expected *Observational Risk* over all tasks.

Algorithm Cost Analysis The most expensive step in BFF is to compute the metric Q_{obs} for each feature of each classification tasks. Suppose there are n tasks with k dimensions each (therefore there are a total of $N = n \times k$ streams), and out of them we have the capacity to observe m

Algorithm 1 The Best Feature First (BFF) Algorithm

inputs: A total of n classification tasks, where each task T_i has k streaming data sources(features).
outputs: Decisions δ_i ($i \in 1, \cdots, n$) for each of the n tasks
static variables: For each of the N streams, one vector $p(x)$ for next distribution, and Markov model K built on data in a sliding window.

1: Compute the predicted distribution $p(x)$ for each feature x, based on the previous $p(x)$ value and the Markov model K.
2: Compute the predicted decision δ_i ($i \in 1, \cdots, n$) for each of the n tasks based $p(x)$ (Equation 3.4).
3: Apply heuristics to prune the set of all features, which results in candidate feature set F_{cand} (see text).
4: For all features $x_j \in F_{cand}$, compute $Q_{Obs}(x_j)$ by Eq 3.9
5: For all features $x_k \notin F_{cand}$, assign $Q_{Obs}(x_k) \leftarrow 0$
6: $observed\_count \leftarrow 0$
7: **while** still data and $observed\_count < Capacity$ **do**
8:     Pick the unobserved stream x_j with the highest $Q_{Obs}(x_j)$ value across all features from all tasks, and observe its actual data value. Break ties randomly.
9:     Update distribution $p(x_j)$ to a unit vector to reflect the observation made, and update the decision δ_i for the task T_i that stream x_j belongs to (Equation 3.4).
10:     Update the Q_{Obs} values for the remaining unobserved streams belonging to task T_i (Equation 3.9).
11:     $observed\_count \leftarrow observed\_count + 1$
12: **end while**
13: Update the Markov model for each stream based on observed values and data expiration from sliding window.

streams. Before any observation, we will perform a total of $O(N)$ computation of Q_{obs} metrics. Then after making each observation, we update metric values for $O(k)$ un-observed features for the observed task, which makes the total Q_{obs} update cost to be $O(m \times k)$. Therefore, each round we perform $[O(N) + O(m \times k)]$ computations of the Q_{obs} metric. We apply some heuristics to avoid evaluating the Q_{obs} of some features. 1) A threshold risk value is adaptively set, (e.g. the 20 percentile of the non-zero Q_{obs} values from the last window) to prune low-risk tasks. 2) We prune features whose Q_{obs} (risk gain) in the last window was below the threshold, and the risk value of the task has changed very little. Thus we avoid features whose observation is not likely to give rise to enough risk gains. Although the worst case is not affected, these heuristics effectively reduce the amortized average computational complexity in our experiments.

To compute the Expected Observational Risk we need to integrate over the entire feature space. To reduce the computational complexity we use *integration by sampling* as validated in [4] and in our own experiments. We omit further details due to length limitations. This sampling is only needed once per time unit. Suppose we obtain h samples on each feature, the total cost of sampling is then $O(h \times N)$, where h is usually a small number (e.g., 10). Maintaining the Markov models for N features each with M distinct values has a $N \times M \times M$ space and time complexity.

5 Experimental Evaluation

Our experiment results indicate that the BFF algorithm outperforms both the random-shedding algorithm and the task-based shedding algorithm LoadStar [4] on multi-source classification tasks, with a reasonable overhead.

Experiment Setups We use the Naïve Bayesian classifier and a 0/1 loss function for risk computation, where the classification error is the percentage of mis-labeled data points. For the Monte Carlo sampling we use 10 sample points for each task.

Synthetic Datasets We generate data for 25 classification tasks each with K features (i.e. K different streaming inputs per task), thus for a total of $25 * K$ input streams. For the experiments shown here K is set to 4. Data are generated for 10000 time units, the first 5000 are used for training and the rest for testing. Due to the independence assumption, the class models on each feature are assigned independently. Half of the K features for each task are assigned with the following class model: $p(x|+) \sim N(0.3, 0.2^2)$, $p(x|-) \sim N(0.7, 0.2^2)$, where $N(\mu, \sigma^2)$ is Normal Distribution with mean μ and variance σ^2. The other half features in each task are assigned with: $p(x|+) \sim N(0.7, 0.2^2)$, $p(x|-) \sim N(0.3, 0.2^2)$. Then the *real* class is assigned based on the joint posterior probability.

For data point movements, we use a random walk model: $x_t = x_{t-1} + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$. Half of the K features in each task are assigned with a σ value of 0.3, and the other half are assigned with a σ value of 0.005. Therefore the features in the same task could have very different movement variances.

Quality of Classification Figure 5 shows the quality of classification under different load shedding percentages for different quality metrics. We use random shedding as the *baseline* for comparison. The horizontal axis shows the percentage of load that is shed from observation, and the vertical axis shows the relative error compared to the error of random-shedding, as follows:

$$\frac{Error_{algorithm}}{Error_{random}}$$

We see that the feature-based greedy algorithm utilizing metric Q_{Bayes} (line C) performs better than the task-based load shedding method LoadStar (line B), while the BFF algorithm (line D), which is feature-based and specifically targeting the Observational Risk, outperforms all other methods. The BFF algorithm achieves more than 45% improvements over random load shedding when the amount of shedding is about 40% to 50%. When the amount of shedding further increases, the improvement drops as prediction becomes less accurate.

Similar experiments were also carried out on real-life datasets for a traffic-jam prediction application, with the same conclusions as above [3].

CPU Cost Savings Because of computational overhead, when we shed x% of data from observation, we actually achieve a total CPU cost saving that is less than x%.

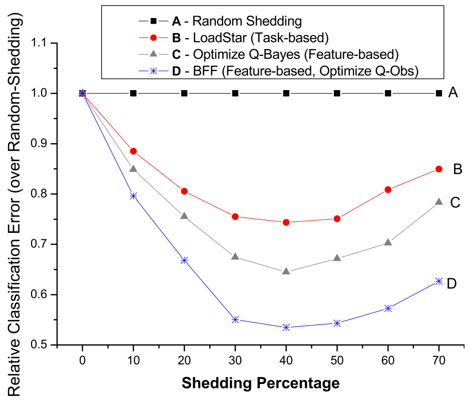


Figure 5: Classification Quality with Load Shedding

Therefore, we measure the total CPU time required under load shedding, and divide it by the total CPU time required without load-shedding. This ratio is then the *effective CPU time saving* achieved by load shedding.

As discussed in Section 1, our algorithm applies to cases when data observation has high costs. In the experiments we assign costs c as in CPU time for observing a data source (for Figure 6, c is 5 msec). This is reasonable in many situations. For example, to extract human faces from video streams as a data pre-processing step, using state-of-the-art technology, it takes 67 msec to recognize faces on a 384x288 image [14]. In comparison, in our experiments it takes about 1 msec to predict a feature value using the Markov model and then classify the task.

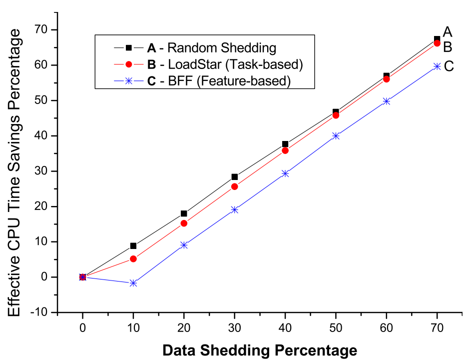


Figure 6: The CPU Cost of Algorithms

Figure 6 shows that LoadStar has an overhead that is a little higher than that of random shedding, but lower than BFF. For the BFF algorithm, the CPU savings from the first 10% tuple shedding is consumed by the algorithm overhead, i.e., a 20% tuple shedding roughly achieves a 10% CPU time saving under this setting.

6 Related Works and Conclusions

LoadStar [4] studies a special case of our problem where every task only has one input stream. Load shedding mechanisms has been studied for Data Stream Management Systems (DSMSs), which generally either employ a random-dropping mechanism [1, 7, 13], rely on user-provided static QoS metric [7], or employ feed-back control theory [8]. These methods do not address the quality requirements of classification tasks. Adapting classifiers for streaming data is another related area [15, 16, 10, 11, 12], which usually studies one-pass incremental algorithms, builds data synopsis, or adapts classifiers to concept-drifts. Our work instead focuses on intelligently dropping, not approximating, input data under overloaded conditions.

In this paper, we adopt a Bayes Risk based approach to optimize the multi-source classification problem in the presence of limited resources. We introduce the notion of Observational Risk as the proper risk measure for feature-based load shedding, and propose the Best Feature First (BFF) algorithm to greedily minimize this risk. Experiments in both synthetic data and real-life data [3] confirms the effectiveness of our algorithm.

References

[1] B. Babcock, M. Datar, and R. Motwani. Load shedding techniques for data stream systems, 2003.

[2] Brian Babcock, Mayur Datar, and Rajeev Motwani. Load shedding for aggregation queries over data streams. In *ICDE*, 2004.

[3] Yijian Bai, Haixun Wang, and Carlo Zaniolo. Load shedding in classifying multi-source streaming data: A Bayes Risk approach. Technical Report TR060027, UCLA CS Department, 2006.

[4] Yun Chi, Philip S. Yu, Haixun Wang, and Richard Muntz. Loadstar: A load shedding scheme for classifying data streams. In *SIAM DM*, 2005.

[5] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience Publication, 2000.

[6] Mohamed M. Gaber et. al. Mining data streams: a review. *SIGMOD Rec.*, 34(2), 2005.

[7] N. Tatbul et. al. Load shedding in a data stream manager. In *VLDB*, 2003.

[8] Yi-Cheng Tu et. al. Control-based quality adaptation in data stream management systems. In *DEXA*, 2005.

[9] Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *ACM SIGMOD Record*, 32(2):5–14, 2003.

[10] Sudipto Guha and Nick Koudas. Approximating a data stream for querying and estimation: Algorithms and performance evaluation. *ICDE*, 2002.

[11] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *KDD*, 2001.

[12] Rouming Jin and Gagan Agrawal. Efficient decision tree construction on streaming data. In *KDD*, 2003.

[13] R. Motwani, J. Widom, A. Arasu, B. Babcock, M. Datar S. Babu, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, approximation, and resource management in a data stream management system.

[14] Paul Viola and Michael J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154, 2004.

[15] Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *SIGKDD*, 2003.

[16] Haixun Wang, Jian Yin, Jian Pei, Philip S. Yu, and Jeffrey X. Yu. Suppressing Model Overfitting in Mining Concept-Drifting Data Streams. In *SIGKDD*, 2006.