Decision Making from Streaming Data

Dissertation submitted in partial fulfilment for the award of the degree

Master of Technology in Computer Science

by

SOUMEN MANDAL

Roll No.: CS2230
M.Tech, 2nd year

Under the supervision of
Dr. Malay Bhattacharyya

Computer and Communication Sciences Division
INDIAN STATISTICAL INSTITUTE

June, 2024

CERTIFICATE

This is to certify that the work presented in this dissertation titled “Decision Making from Streaming Data”, submitted by Soumen Mandal, having the roll number CS2230, has been carried out under my supervision in partial fulfilment for the award of the degree of Master of Technology in Computer Science during the session 2023-24 in the Computer and Communication Sciences Division, Indian Statistical Institute. The contents of this dissertation, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Malay Bhattacharyya
Associate Professor, Machine Intelligence Unit
Associate Member, Centre for Artificial Intelligence and Machine Learning
Associate Member, Technology Innovation Hub on Data Science, Big Data Analytics, and Data Curation
Indian Statistical Institute, Kolkata

Acknowledgements

First and foremost, I want to express my gratitude to *Dr. Malay Bhattacharyya* for all of his help, support, and encouragement during my postgraduate training. His assistance with my dissertation has my sincere gratitude. It was a great experience for me as a new researcher to work with him.

Additionally, I am grateful to *Jyoti Patel* and *Debprasad Kundu* for providing me with the datasets utilized in this dissertation's Chapters 1 and 2. I also thank the academics and researchers at ISI, Kolkata, as well as others who assisted me in conducting the survey that provided the data for Chapter 2.

I am grateful to the CSSC and ISI Library for your multifaceted assistance during my time of need.

My parents, teachers, and friends have supported me through every stage of life, and for that I will always be grateful. Their unwavering support has been essential to me throughout my academic career, particularly with the research I've been doing the last year.

Soumen Mandal

Date: 10-06-2024

Soumen Mandal
Roll No.: CS2230
M.Tech, 2nd year
Indian Statistical Institute

Abstract

In a crowdsourcing environment, judgment analysis involves gathering opinions from a diverse online crowd to reach a consensus. Traditional methods work only when all opinions are available from the start. Our goal is to develop a method for judgment analysis that works as opinions stream in. This dissertation is divided into two parts, each focusing on judgment analysis in a crowdsourcing environment. In the first chapter, we treat all questions and annotators as having equal weight. In the second chapter, we consider different weights for both questions and annotators to make final decisions.We present the first algorithm capable of analyzing crowdsourced opinions in real-time. Tested on two datasets, our method achieves accuracy close to majority voting while requiring only a small amount of space. In the second algorithm We tested it on two datasets, showing it matches the accuracy of majority voting and uses minimal space. This work advances judgment analysis in crowdsourcing, providing a more reliable solution than first for real-time decision-making with online crowdsourced opinions.

Contents

List of Figures

List of Tables

CHAPTER 1

Judgement Anlysis on Opinion Streams based on DGIM Algorithm

1 Judgement Analysis on Opinion Streams

In this chapter, we will discuss about the problem of judgement analysis on opinion streams. In this work, we obtain decisions from the opinions of crowd through judgement analysis. Here, we operate under the assumption that the annotators can only offer ternary opinions, which are represented as 1, 0, or -1 for each question. In this context, 1 signifies a "yes" opinion, 0 indicates a "no" opinion, and -1 represents a "skip" or neutral stand.

1.1 Introduction

Crowdsourcing has become a powerful way to solve many decision-making tasks by using the collective intelligence of many people. It has revolutionized how we tackle problems by harnessing the combined knowledge and skills of a diverse group of individuals [1, 2]. This approach allows organizations and individuals to access a vast pool of resources, ideas, and viewpoints that were previously out of reach [3, 4]. Crowdsourcing promotes collaboration and innovation by letting people from different backgrounds and locations work together towards a common goal.

The capacity of crowdsourcing to produce original and imaginative ideas is one of its primary advantages. By bringing together different perspectives, people can build on each other's ideas, challenge assumptions, and find innovative solutions. Crowdsourcing also allows for efficient use of resources. Instead of relying on a small group of experts, organizations can tap into a much larger talent pool, assigning tasks to those best suited for them. This speeds up the work process and ensures a diverse range of skills and experiences contribute to projects.

Crowd-powered systems have been around for centuries, even though they have gained popularity recently [5]. Amazon Mechanical Turk (MTurk), launched in 2005, is an example of a crowdsourcing platform that allows for scalable and affordable gathering of opinions [6]. Many other platforms have since emerged, like 23andMe, Kickstarter, and Crowdfynd, which can be classified as either collaborative or competitive based on how they use crowd workers.

In competitive crowdsourcing, many people work independently on the same task, competing against each other [7]. In collaborative crowdsourcing, people work together to solve a problem [8]. The first studies on these models were conducted in 2010 and 2011 [9, 10].

Different crowdsourcing platforms use various methods to match tasks to workers. Contest-based platforms like Top-Coder and InnoCentive offer open invitations, rewarding the best submissions [11]. Small tasks are assigned on a first-come, firstserved basis via microtask platforms such as Amazon Mechanical Turk. Effective allocation algorithms are crucial for platforms involving skilled crowds and specialized work [12].

Crowdsourcing platforms can be divided into two models: distributed micro-task and centralized models, using either specialists or non-experts. Tasks like the Netflix Prize and DARPA's Red Balloon competition have shown the effectiveness of expert crowds [7].

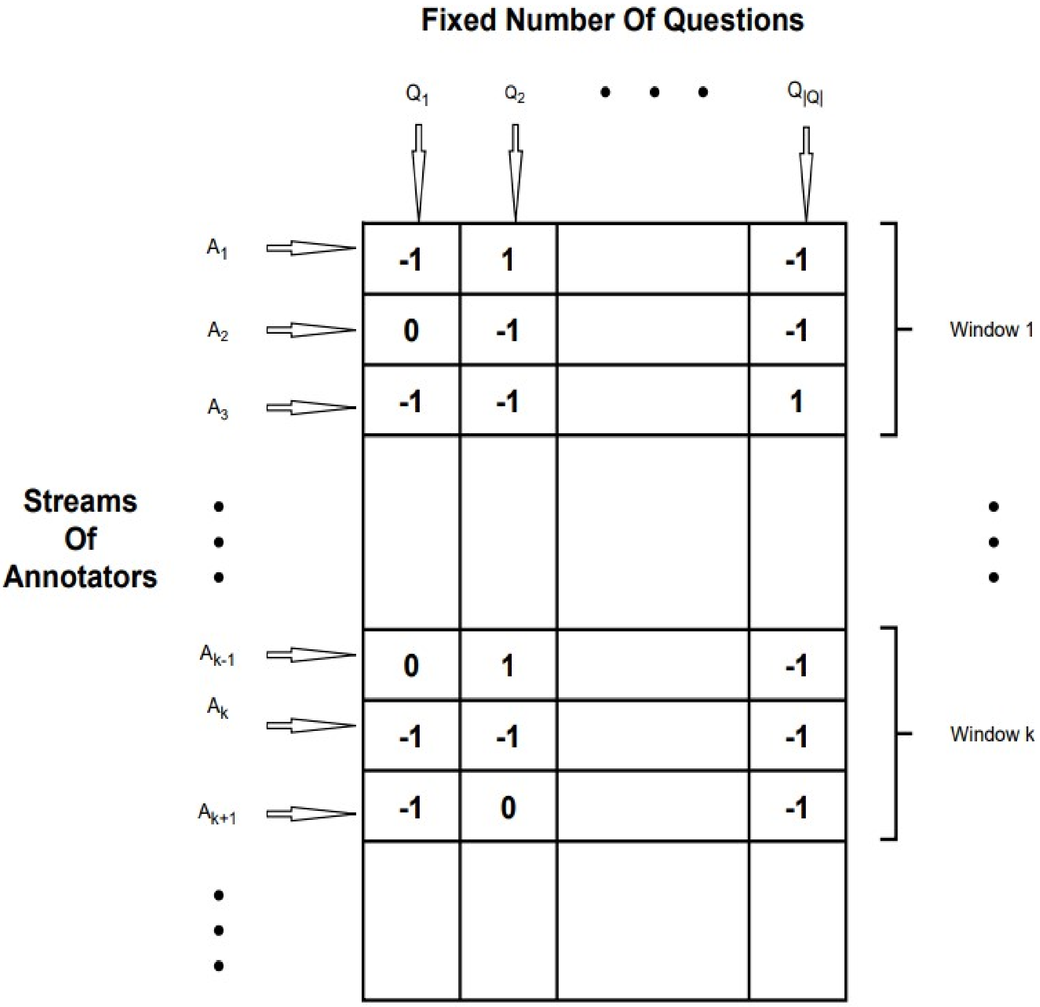
Opinion-based judgments through crowdsourcing have been successful [13, 14]. Multiple opinions can be combined to estimate the "gold" judgment or ground truth for a question [15]. However, involving laypeople often leads to inaccuracies. To make crowdsourcing reliable, it's important to develop algorithms that can separate truth from conflicting and noisy information [14].

With the rise of big data from social networks, online stores, and sensors, we face challenges in efficiently classifying data streams. Unlike traditional models, big data must be analyzed in real-time due to storage limitations [16]. This requires algorithms that can handle data streams effectively, growing sublinearly in storage space and processing time [17].

Judgment analysis has never been done in a streaming scenario. This dissertation proposes a groundbreaking method for performing judgment analysis on data streams.

1.2 Motivation

Traditionally, the problem of judgment analysis for crowdsourced opinions has been approached in a static context . However, the need to analyze continuous data streams is critical for many real-world applications, as it allows for the extraction of valuable insights and the making of swift decisions. Given the vast number of online users engaging in various activities at any moment, it is impractical to gather and process all opinions simultaneously. Crowdsourced feedback typically arrives at different times, making it impossible to store all responses for judgment purposes. To address this, we introduce the first algorithm designed to analyze streaming opinions for judgment analysis while keeping space requirements within a logarithmic bound relative to the number of annotators. While it may seem that modern high-end machines with ample storage could handle this problem, resource-constrained environments present a significant challenge. Additionally, current methods lack scalability, which is a primary theoretical motivation for our work.



	Q_1	Q_2	• • •	$Q_{\|Q\|}$
A_1	-1	1		-1
A_2	0	-1		-1
A_3	-1	-1		1
• • •				
A_{k-1}	0	1		-1
A_k	-1	-1		-1
A_{k+1}	-1	0		-1
• • •				

Figure 1.1: The fundamental framework for judgment analysis applied to streaming opinions. The response matrix maintains a constant number of columns (each representing a specific question), but the number of rows increases as more annotators provide responses over time.

1.3 Problem Formulation

In this section, we define the problem of judgment analysis for streaming data [16]. We consider a set of decision-making questions, $Q = \{q_1, q_2, \dots, q_{|Q|}\}$, and a set of annotators $A = \{A_1, A_2, \dots, A_{|A|}\}$ at a particular timestamp. In a streaming environment, the annotation process is represented by a mapping function $Q \times A \to O$ at a given timestamp, where a window of data is received.

At each timestamp, our goal is to determine the final judgment for all questions in Q based on the received data. For each timestamp, we receive a response matrix R, which is a matrix of size $|A| \times |Q|$. Each element R_{ij} represents the opinion of the i-th annotator for the j-th question, where $R_{ij} \in O$.

In a streaming setting, the data stream $T = \langle T_1, T_2, \dots, T_l \rangle$ consists of continuous opinions. Since we cannot store all opinions at once, we process them using a window-based model, where data is received in chunks. Our aim is to perform judgment analysis on these streams of opinions (response column vector) for each question, ensuring that:

- The processing time for each question is $O(N)$.
- The storage requirement is $O(\text{polylog}(N))$, where N is the number of annotators.

Key points to consider:

- Annotators choose only one option for each question from the set of opinions.
- Annotators do not provide their opinions simultaneously, so responses arrive in a streaming manner.
- Based on the collective opinions, we need to determine the best option for each question.
- Typically, not all annotators respond to every question, making R a sparse matrix.

Our approach is inspired by the DGIM (Datar-Gionis-Indyk-Motwani) algorithm [18].

1.4 Preliminary Details

1.4.1 Basic Terminologies

We define a few key terms in this part that are essential to comprehending judgment analysis. The domain-specific terms used throughout the paper are explained below. Standard terms have their usual meanings unless specified otherwise.

- **Question**: A question is a formulated query that requires a decision or judgment. It is the central element around which the judgment analysis revolves.

- **Annotator**: An annotator is an individual, typically a crowd worker, who provides answers to specific questions. These answers help in making decisions. Annotators may have varying levels of expertise, meaning they might be very good at some aspects of the task but less proficient in others.

- **Opinion**: An opinion is the specific answer given by an annotator to a decision-making question. Opinions can vary widely depending on the annotator's knowledge, experience, and perspective.

- **Annotation**: Annotation refers to the process of collecting these opinions from annotators. It involves gathering diverse responses to the questions posed.

- **Domain of opinions**: This denotes the complete set of possible opinions that can be given for a particular question. It is important to note that this set is finite, meaning there are limited predefined options an annotator can choose from.

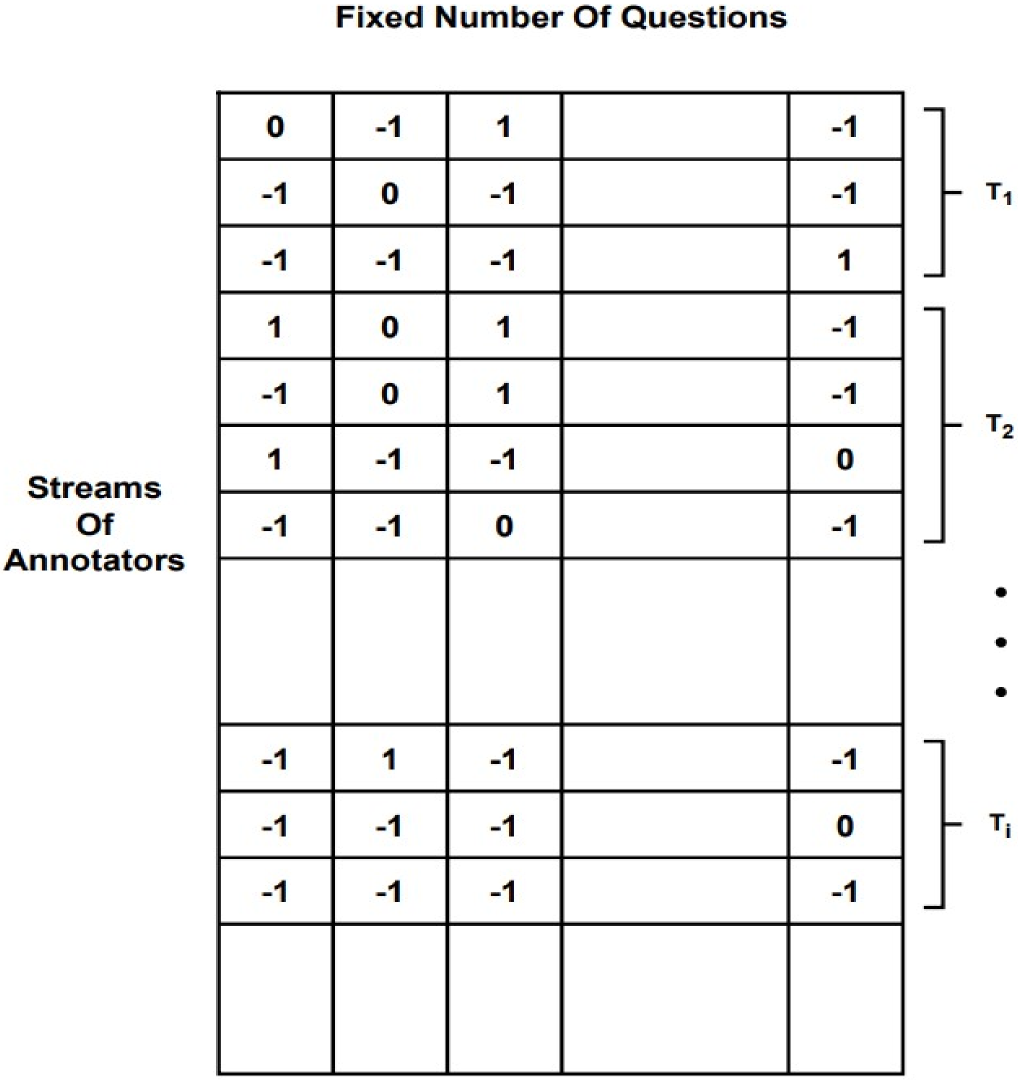
- **Aggregation**: Aggregation is the process of combining the collected opinions from multiple annotators to form a collective judgment for each question. By aggregating these individual opinions, a more accurate and reliable judgment can be obtained.

- **Gold judgment**: The gold judgment is the ground truth or the most accurate opinion for each question. It serves as a benchmark against which other opinions are measured.

- **Question difficulty**: Question difficulty refers to how challenging a particular question is for the annotators to answer. It indicates the level of complexity involved in making a judgment on that question.

- **Annotator accuracy**: Annotator accuracy measures how reliable an annotator is in providing correct judgments. It is determined by how often an annotator's opinions match the gold judgment. Higher accuracy indicates a more dependable annotator.

We use the term *response matrix* to describe the matrix that organizes the opinions collected from a set of annotators (represented by rows) across a set of questions (represented by columns). Each element in this matrix represents the opinion given by an annotator for a specific question. A *response column vector* refers to a single column



0	-1	1		-1
-1	0	-1		-1
-1	-1	-1		1
1	0	1		-1
-1	0	1		-1
1	-1	-1		0
-1	-1	0		-1
-1	1	-1		-1
-1	-1	-1		0
-1	-1	-1		-1

Figure 1.2: This figure shows snapshots of the response matrices at various timestamps. Each row represents the opinions provided by a specific annotator, and each column represents the opinions for a particular question given by different annotators at a particular timestamp. There are three response options: 1 (Yes), 0 (No), and 2 (Skip). A value of $R_{ij} = -1$ indicates that the i^{th} annotator did not provide an opinion for the j^{th} question. Notably, at timestamps T_1 and T_i, there are three annotators, while at timestamp T_2, there are four annotators, demonstrating that the number of annotators varies across different timestamps.

within the response matrix, showing all the responses from different annotators for one particular question, even though many entries might be empty if not all annotators have provided their opinions.

In a streaming environment, these response column vectors are received as continuous streams of data. Therefore, our aim is to analyze multiple streams of opinions (response column vectors) for each question to derive the most accurate judgments. This involves processing the data as it arrives, rather than storing all opinions at once, ensuring efficient and real-time judgment analysis.

1.4.2 Related Work

Majority Voting:

The majority voting algorithm is a straightforward yet powerful method used to combine the opinions or judgments of multiple individuals or classifiers. This method is widely used in fields like data science, machine learning, and decision-making. It works by aggregating the individual judgments or predictions from various sources and determining the final decision based on the most common opinion.

- **Advantages:**
  - *Simplicity*: The algorithm is easy to understand and implement.
  - *Robustness*: It can handle various types of judgments, including binary decisions, categorical predictions, and numerical estimates.
  - *Low Computational Cost*: It is efficient because it does not require complex calculations or optimization.
- **Disadvantages:**
  - *Equal Weighting*: The algorithm assumes that each source's judgment is equally reliable, which may not be true as different sources might have different levels of expertise or accuracy.
  - *Correlated Judgments*: If the judgments from different sources are highly similar, the algorithm may not capture a wide range of perspectives.
  - *Lack of Consensus Resolution*: The algorithm does not provide a way to resolve conflicts or reach a consensus if there is no clear majority, which can lead to ambiguous decisions.

Weighted Majority Voting:

Weighted Majority Voting (WMV) is an enhanced version of the majority voting algorithm. It is used to combine the opinions or judgments of multiple experts or classifiers by assigning weights based on their perceived reliability or expertise. Experts with higher accuracy or more experience are given more influence in the final decision.

- **Advantages:**
  - *Aggregates Diverse Perspectives*: WMV integrates judgments from multiple experts, each offering unique perspectives and approaches.
  - *Handles Expert Heterogeneity*: By assigning different weights based on historical performance, WMV ensures that more reliable experts have greater influence while still considering all contributions.
  - *Improved Accuracy*: Combining judgments from multiple experts can improve overall accuracy by leveraging strengths and compensating for weaknesses.
- **Disadvantages:**
  - *Weight Assignment Challenges*: Determining appropriate weights for experts can be difficult and subjective, potentially leading to biased or suboptimal decisions.
  - *Lack of Error Estimation*: WMV does not explicitly estimate the uncertainty or error of the aggregated decision, making it hard to quantify confidence in the result.
  - *Complexity and Interpretability*: The weighted aggregation process can be complex, making it difficult to understand the influence of each expert's judgment.

1.5 Proposed method

In this section, we first present a simple method for streaming judgment analysis and then describe a more practical and efficient approach.

1.5.1 A Naive Method

In this basic method, we use the average value of the bitstream (average of 0's and 1's) of opinions to determine the final judgment. We begin with the initial bit as the starting mean and continuously update the mean as new bits arrive in the stream. Let

m be the current mean at a given time, i be the new bit, and n be the total number of bits processed up to that point. The updated mean m' is calculated as:

$$m' = \frac{m \cdot (n-1) + i}{n}$$

This formula ensures that the mean is updated incrementally with each new bit, effectively acting as a rolling average that takes into account both the previous mean and the new bit. At any given time, the final judgment can be derived from the current mean value: if the mean is greater than 0.5, the judgment is 1; if it is less than 0.5, the judgment is 0; and if it is exactly 0.5, the result is a tie.

1.5.2 Time Complexity

The time complexity of this method is $O(N)$, where N is the number of bits in the stream for each question. During each iteration, the algorithm performs a constant number of operations to update the mean based on the new bit. Since it processes each bit exactly once, the overall time complexity is linear with respect to the number of bits.

1.5.3 Space Complexity

The space complexity is $O(|A|)$, where $|A|$ is the number of annotators. The algorithm only needs to store a few variables: the current mean, the total number of bits processed, and a loop variable. Regardless of the size of the input stream, the space usage of the algorithm remains constant, making it highly efficient in terms of space.

1.5.4 An Efficient Method

In this method, we keep track of the counts of 0's and 1's in the bitstream of opinions to determine the final judgment. We use buckets to store these counts, where each bucket can hold a varying number of elements, and the sizes of these buckets increase exponentially in reverse order, inspired by the DGIM approach [18]. The counting of 0's and 1's is done in parallel. A bucket represents a segment of the data stream and has the following properties: (i) The size of a bucket (number of 0's or 1's) follows the form 2^i, (ii) Each bucket records the timestamp of its end bit (requiring $O(\log|A|)$ bits) and its size (requiring $O(\log\log|A|)$ bits).

Each bit in the stream is assigned a timestamp using a $(\mod |A|)$ function to map everything within the window. The bitstream is represented as a collection of buckets with the following characteristics:

- There can be one or two buckets of the same size.
- Buckets are sorted by size.
- Buckets do not overlap.
- Buckets are removed if their end-time is more than N time units in the past.

Buckets are updated whenever a new bit arrives. If the oldest bucket's end-time is before N time units from the current time, it is dropped. If the new bit is 0, no changes are needed. If the new bit is 1, do the following: (i) Create a new bucket of size 1 containing the new bit, (ii) Set the timestamp of this new bucket to the current time, and (iii) Starting from $i = 0$, check if there are now three buckets of size 2^i, and if so, combine the oldest two to form a new bucket of size 2^{i+1}.

When combining two buckets into a new one, the timestamp of the newest bit in the old buckets becomes the timestamp of the new bucket. To estimate the number of 0's or 1's in the most recent $k \leq N$ bits: (i) Consider only those buckets whose timestamp is at most k bits in the past, (ii) Sum the sizes of all these buckets except the oldest one, and (iii) Add half the size of the oldest bucket. Some instances of the above process are highlighted in Fig. 1.3.

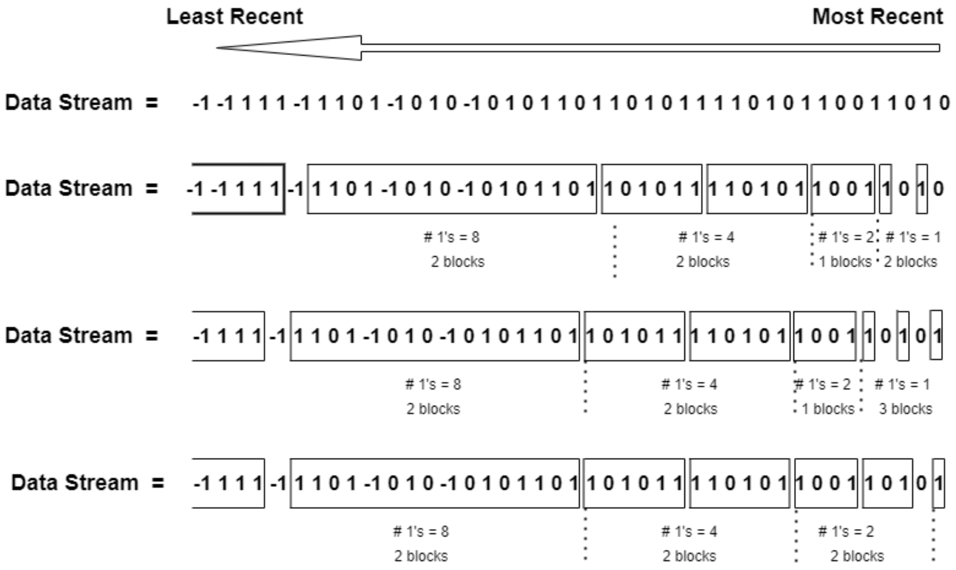

Least Recent
Most Recent
Data Stream = -1 -1 1 1 1 -1 1 1 0 1 -1 0 1 0 -1 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0
Data Stream = -1 -1 1 1 1 -1 1 1 0 1 -1 0 1 0 -1 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0
# 1's = 8
2 blocks
# 1's = 4
2 blocks
# 1's = 2
1 blocks
# 1's = 1
2 blocks
Data Stream = -1 1 1 1 -1 1 1 0 1 -1 0 1 0 -1 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0 1
# 1's = 8
2 blocks
# 1's = 4
2 blocks
# 1's = 2
1 blocks
# 1's = 1
3 blocks
Data Stream = -1 1 1 1 -1 1 1 0 1 -1 0 1 0 -1 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0 1
# 1's = 8
2 blocks
# 1's = 4
2 blocks
# 1's = 2
2 blocks

Figure 1.3: To count the number of 1's in a stream of data that includes 0's, 1's, and -1's, you can use a similar method to count the number of 0's as well.

The main algorithm is presented as Algorithm 1. In this algorithm, we take a window of the response matrix $R_{|A|\times|Q|}$ and return the judgment vector $Judgment$, which is an ensemble of opinions for each question in Q. For each question (steps 1-13), we process a column vector representing the set of opinions given by the annotators and construct buckets to count the number of 0's (steps 2-6) and 1's (steps 7-11). Based on the majority of opinions (step 12), we derive the final judgment and return it (step 14).

Algorithm 1 Main algorithm for opinion ensemble on streaming data

Input: A window of the response matrix $Rm_{|A|\times|Q|}$, where $|A|$ denotes the number of annotators in the current window and $|Q|$ denotes the number of questions.
Output: The judgment vector $Judgment$ consists of an ensemble of opinions for each question in Q.
Algorithmic Steps:

```
 1: for $i = 1$ to $|Q|$ do
 2:     $target \leftarrow 0$
 3:     $TS \leftarrow 0$
 4:     $BUCKETS \leftarrow$ INITIAL-BUCKETING($|A|$)
 5:     RUN($Rm[*][i]$, $|A|$, $TS$, $BUCKETS$, $target$)
 6:     $count0 \leftarrow$ COUNT($|A|$, $TS$, $BUCKETS$)          ▷ Count of 0's
 7:     $target \leftarrow 1$
 8:     $TS \leftarrow 0$
 9:     $BUCKETS \leftarrow$ INITIAL-BUCKETING($|A|$)
10:     RUN($Rm[*][i]$, $|A|$, $TS$, $BUCKETS$, $target$)
11:     $count1 \leftarrow$ COUNT($|A|$, $TS$, $BUCKETS$)          ▷ Count of 1's
12:     $Judgment[i] \leftarrow \max(count0, count1)$
13: end for
14: return $Judgment$
```

The *INITIAL-BUCKETING* procedure initializes a set of buckets based on a given parameter a. It creates a series of empty buckets represented by the variable $BUCKETS$. The number of buckets is determined by the logarithm base 2 of a, with each bucket initially empty. Finally, it returns the set of initialized buckets, preparing them for further data organization and processing based on the specified parameter a.

```
INITIAL-BUCKETING($a$)
{
for $i := 1$ to $\lfloor \log_2(a) \rfloor$ do
    $BUCKETS[i] \leftarrow \emptyset$
end for
return $BUCKETS$
}
```

Table 1: The INITIAL-BUCKETING procedure initializes buckets based on the given parameter a.

```
OLDBUCKET($a$, $TS$, $BUCKETS$)
{
obIndex ← 0                                  ▷ Initialize old bucket index
obTS ← 0                                     ▷ Initialize old bucket timestamp
for $i := 1$ to $size(BUCKETS)$ do           ▷ Each bucket in $BUCKETS$
    for $ets \in BUCKETS[i]$ do              ▷ Each element in $BUCKETS[i]$
        if $ets \geq (TS - a)$ then
            $obIndex \leftarrow i$
            $obTS \leftarrow ets$
        else
            return $obIndex, obTS$
        end if
    end for
    return $obIndex, obTS$
end for
}
```

Table 2: The OLDBUCKET procedure identifies the oldest relevant bucket in the data structure based on the parameters a and TS.

The *OLDBUCKET* procedure identifies the oldest bucket within a specified time window, determined by a and TS, in a structure called $BUCKETS$. It initializes $obIndex$ and $obTS$ to 0, then iterates through each bucket and its elements. If an element's timestamp is within the time window $(TS - a)$, it updates $obIndex$ and $obTS$. If no such element is found in a bucket, it returns the current $obIndex$ and $obTS$, effectively locating the oldest relevant bucket in the data structure.

The *UPDATE* method adjusts the buckets based on certain rules. If there are more than two buckets of the same size, they are merged. The function *delete*(X, i) removes the element at index i from X and returns it. The function *insert*(X, y, i) inserts the element y at index i of X and shifts the subsequent elements one position forward.

The *COUNT* method computes a count using the parameters a, TS, and $BUCKETS$. It initializes a count variable to zero, finds the oldest bucket using the *OLDBUCKET* function, and then iterates through the buckets. Depending on the timestamp conditions, it increments the count accordingly.

The *RUN* method processes a stream of data using the parameters C, a, TS, $BUCKETS$, and $target$. It iterates through each data point, increments the timestamp, checks the oldest bucket, and updates the buckets if the data point matches the target. It takes no action if the data point is -1.

```
UPDATE($BUCKETS$)
{
for $i := 1$ to $size(BUCKETS)$ do                    ▷ Each bucket in $BUCKETS$
    $l \leftarrow length(BUCKETS[i])$
    if $l \geq 2$ then
        $delete(BUCKETS[i], l-1)$                     ▷ Remove the last element
        $temp \leftarrow delete(BUCKETS[i], l-2)$     ▷ Remove the second last element
        if $i \neq size(BUCKETS) - 1$ then
            $insert(BUCKETS[i], temp, 0)$             ▷ Insert at the beginning
        end if
    end if
end for
return $BUCKETS$
}
```

Table 3: The UPDATE procedure merges buckets if there are more than two of the same size.

```
COUNT($a$, $TS$, $BUCKETS$)
{
$count \leftarrow 0$
$obIndex, obTS \leftarrow$ OLDBUCKET($a$, $TS$, $BUCKETS$)
for $i := 1$ to $size(BUCKETS)$ do
    if $i \geq obTS$ then
        return $count + 1$
    end if
    for $endTS \in BUCKETS[i]$ do
        if $endTS \geq obTS$ then
            $count \leftarrow count + 2^i$
        else if $endTS == obTS$ then
            $count \leftarrow count + 0.5 * 2^i$
        end if
    end for
end for
return $count + 1$
}
```

Table 4: The COUNT procedure calculates the count based on timestamps and bucket conditions.

```
RUN($C$, $a$, $TS$, $BUCKETS$, $target$)
{
for each $c \in C$ do                         ▷ Process each data point $(0/1/-1)$ in the stream
    $TS \leftarrow TS + 1$
    $obIndex, obTS \leftarrow OLDBUCKET$(a,TS,BUCKETS)
    if $obTS \neq 0$ and $obTS == TS - a$ then
        if $obTS \in BUCKETS[obIndex]$ then
            $BUCKETS[obIndex] \leftarrow BUCKETS[obIndex] - obTS$
        end if
    else if $c == target$ then                ▷ Current data matches the target $(0/1)$
        $BUCKETS[0] \leftarrow \{TS\} \cup BUCKETS[0]$
        $BUCKETS \leftarrow UPDATE(BUCKETS)$
    end if                                    ▷ Take no action if the data point is -1
end for
}
```

Table 5: The RUN procedure processes each data point in the stream and updates the buckets accordingly.

1.5.5 Time Complexity

Algorithm 1 processes each question in the input data by running a loop $|Q|$ times, where $|Q|$ is the number of questions. For each question, the initial steps (2-3) take constant time. The call to `INITIAL-BUCKETING()` (step 4) involves a loop that runs $\lfloor \log_2 |A| \rfloor$ times, where $|A|$ is the number of annotators, resulting in a time complexity of $O(\log |A|)$. The `RUN()` function (step 5) has a time complexity of $O(|A|)$. The `COUNT()` function (step 6) iterates through all the buckets, leading to a time complexity of $O(\log |A|)$. This process (steps 2-6) for counting the number of 0's is repeated (steps 7-11) for counting the number of 1's. The final judgment is computed in constant time as it only involves comparing two values. Thus, the total time complexity is:

$$|Q| \times \left(\underbrace{O(\log |A|) + O(|A|) + O(\log |A|)}_{\text{steps 2-6}} + \underbrace{O(\log |A|) + O(|A|) + O(\log |A|)}_{\text{steps 7-11}} \right)$$

$\simeq O(|Q||A|)$.

1.5.6 Space Complexity

Algorithm 1 processes each question by running a loop $|Q|$ times, where $|Q|$ is the number of questions. For each question, `INITIAL-BUCKETING()` (step 4) initializes $\log |A|$ buckets, each with a constant length, k. Therefore, the total space required for managing the `BUCKETS` is $k \cdot \log |A|$. This space is reused in the other function calls to `RUN()`

(step 5) and `COUNT()` (step 6). The space required for the other variables is constant, denoted as v. The counting process (steps 2-6) for the number of 0's is repeated (steps 7-11) for the number of 1's. The final judgment calculation also uses a constant space, denoted as j. Thus, the total space complexity is:

$$|Q| \times \left(\underbrace{k \cdot \log|A| + v}_{\text{steps 2-6}} + \underbrace{k \cdot \log|A| + v}_{\text{steps 7-11}} + \underbrace{j}_{\text{step 12}} \right)$$

$\simeq O(|Q| \log|A|)$.

The overall space complexity of the algorithm is the sum of the space complexity for `BUCKETS` and the space complexity for other variables:

Overall Space Complexity = Space complexity for BUCKETS+Space complexity for other variabl

$$\text{Overall Space Complexity} = O(\log|A|) \times C + D.$$

1.5.7 Error Factor

Since there is at least one bucket of each of the sizes less than 2^i, and at least one from the oldest bucket, the true sum is no less than 2^i. Thus, the error is at most 50%.

1.6 Empirical Analysis

We used two datasets, a large-scale dataset called Fact Evaluation and a small-scale dataset named WVSCM, to evaluate the performance of our suggested strategy in comparison to current techniques. On a system with an 11th Gen Intel(R) Core(TM) i5-1135G7 CPU running at 2.4 GHz, 16 GB of RAM (15.8 GB usable), and a 64-bit operating system, we carried out the experiments using Python 3.0.

1.6.1 Results on WVSCM Dataset

The WVSCM dataset includes data from an experiment on Duchenne smiles, studied by Whitehill et al. [19]. The task for the Mechanical Turk workers was to label facial images as either Duchenne or Non-Duchenne smiles. A Duchenne smile, also known as an "enjoyment" smile, can be distinguished from a Non-Duchenne smile, also called a "social" smile, by the activation of the Orbicularis Oculi muscle around the eyes. This distinction is important in fields such as psychology, human-computer interaction, and marketing research. Even experts trained in the Facial Action Coding System find it difficult to accurately identify Duchenne smiles.

Table 6: Performance in terms of accuracy for the WVSCM dataset. The highest accuracy in each column is highlighted in bold.

Algorithm	Accuracy(%)	Data Setting
Proposed Efficient Method	**83.87**	Streaming
Majority Voting	75.00	Non-Streaming
Weighted Majority Voting	75.62	Non-Streaming

In the ground truth file, each row contains two columns: the image ID and the Duchenne label (0 or 1). Out of the 160 images, 58 showed Duchenne smiles. There are 64 annotators who provided their opinions. Among these annotators, 14 had an accuracy of 50% or higher, 13 had an accuracy of 60% or higher, 7 had an accuracy of 70% or higher, and only 1 had an accuracy of 80% or higher. In the response file, each row has three columns: the image ID, the labeler ID, and the label (0 for non-Duchenne, 1 for Duchenne). Missing values were discarded to improve the dataset's reliability, and entries where annotators skipped the evaluation were initially labeled as "-1". Keeping skipped values as "-1" acknowledges instances where annotators refrained from making specific judgments. This refined dataset allows for clearer analysis of the presence or absence of Duchenne smiles. This pre-processing ensures consistent binary classification, making the dataset ready for analysis.

We compared our algorithm with the well-known majority voting and weighted majority voting algorithms on the WVSCM dataset. Table 6 shows the comparative accuracy values. Our method, based on a streaming setting, achieved a slightly lower accuracy. However, this difference is not statistically significant. Moreover, our space requirements are limited by a logarithmic factor of the number of annotators, unlike the other methods, which are not suitable for a streaming setting.

1.6.2 Results on Fact Evaluation Dataset

The Fact Evaluation dataset includes judgments about public figures on Wikipedia, specifically focusing on whether individuals "attended or graduated from an institution." This dataset contains 42,623 examples, each evaluated by at least 5 annotators, resulting in a total of 216,725 judgments from 57 trained annotators.

Each data entry in the full version of the dataset is represented as a triplet: the relation (predicate), the subject of the relation, and the object of the relation. Subjects and objects are identified by their Freebase MIDs, and the relation is a Freebase property. Evidence supporting these relations is provided as URLs and excerpts from web pages judged by the annotators. The data is in JSON format, with each line containing

Table 7: Performance in terms of accuracy for the Fact Evaluation dataset. The highest accuracy in each column is highlighted in bold.

Algorithm	Accuracy(%)	Data Setting
Proposed Efficient Method	80.87	Streaming
Majority Voting	94.54	Non-Streaming
Weighted Majority Voting	**95.82**	Non-Streaming

fields such as predicate, subject, object, evidence, web page, supporting text, judgments from human annotators, annotator identity hash code, and annotator judgment (0/1/2 for no/yes/skip, respectively).

To improve clarity, the original "2" annotation for "Skip" has been changed to "-1." Thus, the refined scale now uses "0" for "No," "1" for "Yes," and "-1" for "Skip," making it easier to interpret the annotations. This consistent labeling system simplifies the analysis of the data. The dataset includes answers to 576 facts as the gold standard. The basic version of the dataset has two columns: question ID and metadata. The metadata contains a JSON-encoded dictionary with the judgments of all raters for each question, along with other relevant data. This dataset is provided under the Creative Commons license.

We compared our algorithm with the majority voting and weighted majority voting algorithms on the Fact Evaluation dataset. Table 7 shows the comparative accuracy values. Our method, which operates in a streaming setting, achieved lower accuracy because of the large size of the data. However, our method's space requirements are limited by a logarithmic factor of the number of annotators, unlike other methods which are not suitable for a streaming setting.

1.7 Conclusion and Future Work

We have introduced a space-efficient method for judgment analysis in a streaming context. Since the processes for counting 0's and 1's are independent, they can be executed concurrently. Additionally, the algorithm can handle each question in parallel. Therefore, using parallel computing can greatly reduce the time required by our algorithm. Although the error margin in our approach could be minimized further, this would require additional space. Our method effectively counts 0's and 1's from a stream of opinions, which can be used to calculate the entropy of these opinions. This entropy measurement can help quantify the difficulty of a question. In practical terms, higher entropy indicates greater variability in opinions, suggesting a more challenging question.

Moreover, our approach's flexibility in handling large-scale datasets in a streaming setting demonstrates its potential for real-world applications where data is continuously generated. The ability to process data on-the-fly without significant space requirements makes it suitable for use in dynamic environments, such as online surveys, live feedback systems, and real-time data analysis in social media platforms. Future work could focus on optimizing the trade-off between space efficiency and error reduction, as well as exploring the integration of this method with other machine learning algorithms to enhance its robustness and applicability.

This insight could be used to develop a weighted version of our approach for use in streaming scenarios. By assigning weights based on the entropy of opinions, we can better account for the varying difficulty of questions, leading to more accurate and reliable judgments.

CHAPTER 2

Weighted Judgement Analysis on Opinion Streams

2 Weighted Judgement Analysis on Opinion Streams

In the previous chapter, we analyzed opinions without considering the reliability of different annotators and the difficulty of the questions. However, to achieve more accurate results, we need to account for both the annotator's reliability and the difficulty of the questions.

2.1 Introduction

Crowdsourcing has become a powerful tool, using the collective knowledge and skills of many people to solve various tasks on a large scale. By tapping into the insights and abilities of a diverse group, crowdsourcing has changed the way we solve problems, innovate, and collaborate in our connected world. It allows organizations and individuals to access a wide range of ideas and perspectives that were previously out of reach [3, 4].

In decision-making, crowdsourcing can create new and innovative solutions by combining different viewpoints and ideas. This collective intelligence often leads to creative thinking and groundbreaking discoveries that traditional methods might miss [8]. Crowdsourcing also helps allocate resources efficiently by connecting tasks with the people who have the right skills to complete them [12].

While crowdsourcing isn't new, advances in technology and the rise of online platforms have made it more popular [6]. Platforms like Amazon Mechanical Turk (MTurk) have made it easier and more affordable to access human intelligence for tasks such as information gathering and opinion collection [10]. However, as these platforms evolve, there is a growing need for effective algorithms to manage skilled crowds and specialized tasks [20].

One important area for these algorithms is judgment analysis, where multiple opinions are combined to determine the correct answers to various questions [15]. Traditional crowdsourcing models often struggle with accuracy due to noise and conflicting opinions from non-experts [13]. To address these issues, we need algorithms that can identify the truth in noisy crowdsourced data, especially in scenarios involving continuous data streams [14].

The influx of data from social networks, sensors, and online platforms has created new challenges for decision-making and analysis [17]. Traditional data analysis methods are not equipped to handle the speed and volume of streaming data, prompting the need for new algorithms designed for this environment [16]. The Streamed Weighted Majority Voting algorithm offers a promising solution for real-time judgment analysis, providing efficient and accurate aggregation of opinions in dynamic data streams.

Previously, judgment analysis for crowdsourced opinions was done using static methods. However, with the increasing prevalence of streaming data, it's essential to adapt these methods for continuous data streams. This shift to streaming data analysis is both necessary and challenging.

In this chapter, we propose a new method for analyzing judgments in streaming data, using weighted majority voting to achieve reliable results in changing data streams.

This introduction highlights the importance of crowdsourcing, the challenges it faces, and the need for innovative algorithms like the Streamed Weighted Majority Voting algorithm to handle these challenges in streaming data analysis.

2.2 Preliminary Details

2.2.1 Basic Terminologies

Here we present some more basic terminology related to this chapter streamed weighted majority voting.

- **Annotator Weight**: Not all annotators are equally reliable. Some may consistently provide accurate answers, while others may not. By assigning weights to annotators based on their past performance, we can give more importance to the opinions of reliable annotators.

- **Question Difficulty**: Some questions are harder to answer correctly than others. By considering the difficulty of each question, we can adjust our expectations and calculations accordingly. For example, a correct answer to a difficult question might be more significant than a correct answer to an easy one.

- **Entropy**: Entropy is a measure of uncertainty or randomness. In the context of judgment analysis, it is used to quantify the difficulty of a question based on the distribution of responses.

2.3 Proposed Method

In the realm of increasing streaming data, the real-time analysis of annotator judgments is paramount for applications such as sentiment analysis and opinion mining. This methodology introduces the Streamed Weighted Majority Voting algorithm (SWM algorithm), a novel approach aiming to provide an accurate and dynamically adapting ensemble of opinions from annotators as streaming data unfolds.

2.3.1 Prposoed Algorithm

The Streamed Weighted Majority Voting (SWM) algorithm starts by accepting key input parameters. These include the `response_matrix`, which captures annotator responses, and lists `lst_0bit` and `lst_1bit`, which count the 0 and 1 bits for each column. The `acc` parameter represents annotator accuracies, and `lr` is the learning rate for dynamic updates.

The SWM algorithm combines weighted majority voting and dynamic learning to analyze annotator responses in a streaming data window. It focuses on annotator accuracy, question difficulty, and a learning rate to adapt over time.

First, the initial question difficulty is calculated using the entropy of 0 and 1 bits for each column. The variables `a` and `b` are set to mark the start and end of the streaming window. The algorithm sets up a matrix `M` to store weighted opinions and initializes the `Wmr` list. Annotator accuracies are initialized with given values. The algorithm processes the streaming data by updating question difficulty with each new window of data. For each column in the window, the DGIM algorithm counts the 0 and 1 bits, updating `lst_0bit_new` and `lst_1bit_new`. The SWM algorithm then updates matrix `M` with weighted opinions based on the current window and annotator accuracies. The weighted majority result is calculated, and annotator accuracies are updated dynamically using this result and the question difficulty, incorporating the learning rate.

This process continues, with the streaming window moving forward until all the data is processed. The algorithm's performance is measured by comparing the weighted majority result to ground truth labels, providing insights into its real-time adaptability and accuracy.

The swm() subroutine implements a dynamic updating mechanism for weighted majority voting in the context of streamed data. It calculates the entropy of each column in the current window of responses, assigning higher importance to columns with lower entropy. Using the provided accuracy of annotators and a learning rate, it updates the weighted majority matrix accordingly, favoring more accurate and less uncertain columns. This adaptive approach allows for real-time adjustments to the model based on incoming data, enhancing its robustness and accuracy in processing streaming data.

The function `calc_q_entropy` takes lists of counts for 0's and 1's in each question. It calculates the uncertainty or randomness of each question's responses by finding the proportion of 1s, then computing the entropy. Finally, it returns a list of entropy values for each question.

The calculate_entropy function computes the entropy of a given probability distribution. If the probability is either 0 or 1, indicating absolute certainty, the entropy

Algorithm 2 Main algorithm for weighted judgment analysis on streaming data

Input: Matrix Rm with shape (r, c) containing annotator responses, List $L0b$ containing counts of 0 bits for each column, List $L1b$ containing counts of 1 bits for each column, List acc representing annotator accuracies, Float lr representing the learning rate and also import numpy Python library as np.
Output: The judgment vector Jd consists of weighted ensemble of opinions corresponding to each question in Q.

```
# Initialization
$Qd \leftarrow$ calc_q_entropy($L0b$, $L1b$)
$a \leftarrow 0$
$b \leftarrow 50$
$M \leftarrow$ init_matrix($p$, $r$, $c$, $-1$)
$Wmr \leftarrow$ init_list($r$, $-1$)
$annotator\_accuracy \leftarrow$ [1] * Rm.columns
while $b \leq Rm.rows$ do
    $A \leftarrow$ get_window($Rm$, $a$, $b$)
    # Update question difficulty based on the new window
    set $L0b\_new$ to empty List
    set $L1b\_new$ to empty List
    for $i$ from 0 to $Rm.columns - 1$ do
        $target \leftarrow 0$
        $TS \leftarrow 0$
        $BUCKETS \leftarrow$ INITIAL-BUCKETING($|A|$)
        RUN($Rm[a, b][i]$, $|A|$, $TS$, $BUCKETS$, $target$)
        $res\_0b \leftarrow$ COUNT($|A|$, $TS$, $BUCKETS$)
        $L0b\_new$.Append($res\_0b$)
        $target \leftarrow 1$
        $TS \leftarrow 0$
        $BUCKETS \leftarrow$ INITIAL-BUCKETING($|A|$)
        RUN($Rm[a, b][i]$, $|A|$, $TS$, $BUCKETS$, $target$)
        $res\_1b \leftarrow$ COUNT($|A|$, $TS$, $BUCKETS$)
        $L1b\_new$.Append($res\_1b$)
    end for
    $M \leftarrow$ swm($M$, $A$, $a$, $annotator\_accuracy$, $L1b\_new$, $L0b\_new$, $lr$)
    $Wmr \leftarrow$ argmax($M$, axis=1)
    # Calculate new question difficulty
    $Qd \leftarrow$ calc_q_entropy($L0b\_new$, $L1b\_new$)
    $acc \leftarrow$ update_acc($acc$, $Wmr$, $A$)
    $a \leftarrow b$
    $b \leftarrow b + 50$
end while
return $Wmr$
```

Algorithm 3 swm() Subroutine

Input: Matrix J representing current weighted majority, Matrix $block$ representing the current window of responses, Integer a representing the starting index of the window, List $accuracy$ representing accuracy of annotators, Lists lst_for_1bit and lst_for_0bit representing counts of 1s and 0s for each column, Float $learning\_rate$ representing the learning rate.
Output: Updated matrix J after applying streamed weighted majority.

```
$entropy\_lst \leftarrow$ calc_q_entropy($lst\_for\_1bit$, $lst\_for\_0bit$)
for $j$ from 0 to number of columns in block $-1$ do
    $column\_entropy \leftarrow entropy\_lst[j]$
    $importance\_factor \leftarrow \exp(-column\_entropy) \cdot learning\_rate$
    for $k$ from 0 to number of rows in block $-1$ do
        if $block[k][j] \neq -1$ then
            $J[j][block[k][j]] \mathrel{+}= accuracy[a+k] \cdot importance\_factor$
        end if
    end for
end for
return $J$
```

Algorithm 4 calc_q_entropy() Subroutine

Input: List $L0b$ representing counts of 0s for each column, List $L1b$ representing counts of 1s for each column
Output: List $lst\_for\_ques\_diff$ representing question difficulties for each column

```
for $i$ from 0 to length($L0b$) − 1 do
    $prob \leftarrow$ calculate_probability($L0b[i], L1b[i]$)
    $entropy \leftarrow$ calculate_entropy($prob$)
    Append $entropy$ to $lst\_for\_ques\_diff$
end for
return $lst\_for\_ques\_diff$
```

is considered to be 0. Otherwise, it calculates the entropy using the formula for Shannon's entropy and returns the result. This function quantifies the amount of uncertainty or randomness in a probability distribution.

Algorithm 5 Calculate_Entropy() Subroutine

Input: Float *prob* representing a probability value
Output: Float representing the calculated entropy

```
if prob = 0 or prob = 1 then
    return 0
else
    return $-(\text{prob} \cdot \log_2(\text{prob}) + (1 - \text{prob}) \cdot \log_2(1 - \text{prob}))$
end if
```

The function update_accuracy(update_acc) adjusts the accuracy scores of annotators based on their responses to a window of data compared to the aggregated result of that window. It iterates over each annotator's response, updating their accuracy score. If an annotator's response matches the window result, their accuracy score is incremented slightly. Otherwise, it's decremented. The function returns the updated accuracy scores. This mechanism helps dynamically adjust annotators' reliability based on their performance over time.

Algorithm 6 update_acc() Subroutine

Input: List *accuracy* representing annotator accuracies, List *window_result* representing the results from the current window, List *annotator_responses* representing annotator responses
Output: Updated list *accuracy* after applying adjustments

```
for i from 0 to length(annotator_responses) − 1 do
    if not array_equal(annotator_responses[i], −1) then
        if array_equal(annotator_responses[i], window_result[i]) then
            accuracy[i] += 0.1
        else
            accuracy[i] −= 0.1
        end if
    end if
end for
return accuracy
```

2.3.2 Time Complexity

Algorithm 2 processes each question in the input data by running a loop $|Q|$ times, where $|Q|$ is the number of questions. For the overall time complexity of the main algorithm:

- Initialization steps:
  - `calc_q_entropy(L0b, L1b)`: $O(c)$
  - `init_matrix(p, r, c, -1)`: $O(prc)$
  - `init_list(r, -1)`: $O(r)$
  - Initializing `annotator_accuracy`: $O(c)$
- While loop runs $\frac{n}{50}$ times (assuming $Rm.rows = n$):
  - `get_window(Rm, a, b)`: $O(c)$
  - Processing each column within the loop:
    - `INITIAL-BUCKETING(|A|)`: $O(\log|A|)$
    - `RUN(Rm[a,b][i], |A|, TS, BUCKETS, target)`: $O(|A|)$
    - `COUNT(|A|, TS, BUCKETS)`: $O(\log|A|)$
    - Total for each column: $O(|A|)$
  - `swm(M, A, a, annotator_accuracy, L1b_new, L0b_new, lr)`: $O(rc)$
  - `argmax(M, axis=1)`: $O(rc)$
  - `calc_q_entropy(L0b_new, L1b_new)`: $O(c)$
  - `update_acc(acc, Wmr, A)`: $O(r)$

  Combining these, the overall time complexity per while loop iteration is:

  $$O(c) + O(rc) + O(c) + O(rc) + O(r) = O(rc)$$

  Thus, the total time complexity for the while loop is:

  $$\frac{n}{50} \times O(rc) = O(nrc)$$

2.3.3 Space Complexity

For each question, `INITIAL-BUCKETING()` (line 21) initializes $\log|A|$ buckets, each with a constant length, k. Therefore, the total space required for managing the `BUCKETS` is

$k \cdot \log|A|$. This space is reused in the other function calls to `RUN()` (line 22) and `COUNT()` (line 23). The space required for the other variables is constant, denoted as v. The counting process (lines 21-25) for the number of 0's is repeated (lines 27-31) for the number of 1's. The final judgment calculation also uses a constant space, denoted as j. Thus, the total space complexity is:

$$|Q| \times \left(\underbrace{k \cdot \log|A| + v}_{\text{lines 21-25}} + \underbrace{k \cdot \log|A| + v}_{\text{lines 27-31}} + \underbrace{j}_{\text{final judgment}} \right)$$

$\simeq O(|Q|\log|A|)$.

The overall space complexity of the algorithm is the sum of the space complexity for `BUCKETS` and the space complexity for other variables:

$$\text{Overall Space Complexity} = \text{Space complexity for BUCKETS} + \text{Space complexity for other variabl}$$

$$\text{Overall Space Complexity} = O(|Q|\log|A|) + O(|Q|).$$

Considering the matrix and other variables, the space complexity becomes:

$$O(prc) + O(rc) + O(c) + O(r)$$

Thus, the final space complexity is:

$$O(prc)$$

2.3.4 Summary

- Time Complexity: $O(nrc)$
- Space Complexity: $O(prc)$

2.4 Empirical Analysis

We used three datasets, two large-scale dataset called Fact Evaluation and Sentiment Analysis Dataset and a small-scale dataset named WVSCM, to evaluate the effectiveness of our suggested methodology in comparison to current methods. On a system with an 11th Gen Intel(R) Core(TM) i5-1135G7 CPU running at 2.4 GHz, 16 GB of RAM

(15.8 GB usable), and a 64-bit operating system, we carried out the experiments using Python 3.0.

2.4.1 Results on WVSCM Dataset

The dataset details are already given in the previous chapter. Hence, the results of weighted judgement analysis is only provided below.

Table 8: Performance in terms of accuracy for the WVSCM dataset. The highest accuracy in each column is highlighted in bold.

Algorithm	**Accuracy(%)**	**Data Setting**
Proposed Efficient Method	75.00	Streaming
Majority Voting	75.00	Non-Streaming
Weighted Majority Voting	**75.62**	Non-Streaming

2.4.2 Results on Fact Evaluation Dataset

The dataset details are already given in the previous chapter. Hence, the results of weighted judgement analysis is only provided below.

Table 9: Performance in terms of accuracy for the Fact Evaluation dataset. The highest accuracy in each column is highlighted in bold.

Algorithm	**Accuracy(%)**	**Data Setting**
Proposed Efficient Method	94.73	Streaming
Majority Voting	94.54	Non-Streaming
Weighted Majority Voting	**95.82**	Non-Streaming

2.4.3 Results on Sentiment Analysis Dataset

We used a sentiment analysis dataset provided by the crowd-powered company CrowdFlower. This dataset asks annotators to judge the sentiment of tweets about the weather. It has 98,979 tweets in all, with at least five annotators evaluating each one, for a total of roughly 569,375 answers.

The dataset includes detailed information such as:

- **Question ID**: The ID of the tweet being evaluated.
- **Rater ID**: The ID of the annotator.
- **Judgment**: The annotator's answer, which can be one of the following:
  - `0`: Negative
  - `1`: Neutral (the author is just sharing information)
  - `2`: Positive
  - `3`: Tweet not related to the weather
  - `4`: Cannot determine the sentiment
- **Tweet Text**: The content of the tweet.
- **Country, Region, City**: The location details of the annotator.
- **Started At**: The time when the annotator started the evaluation.
- **Created At**: The time when the annotator finished the evaluation.

For our analysis, we used a simpler version of this dataset with just three columns:

- **Question ID**: The ID of the tweet.
- **Rater ID**: The ID of the annotator.
- **Judgment**: The annotator's answer (0 to 4).

In our experiment, we converted the judgments as follows:

- `0` (Negative) remains `0`.
- `2` (Positive) becomes `1`.
- `4` (Cannot determine) becomes `-1`.

We ignored the other responses (`1` and `3`) in our analysis.

Table 10: The performance in terms of accuracy obtained for the Sentiment Analysis dataset. The best accuracy over the column is shown in bold.

Algorithm	**Accuracy(%)**	**Data Setting**
Proposed Efficient Method	94.26	Streaming
Majority Voting	90.67	Non-Streaming
Weighted Majority Voting	**97.12**	Non-Streaming

2.5 Conclusion and Future Work

We have proposed a space-efficient approach to judgment analysis in a streaming setting.The provided code implements a dynamic sentiment analysis framework that adapts over time by incorporating streamed weighted majority voting. The utilization of the Extended DGIM algorithm efficiently handles the processing of binary responses within a sliding window, contributing to the real-time nature of the approach. Annotator accuracy and question difficulty are dynamically updated based on incoming responses, allowing the system to learn and adjust continuously. Here, the learning rate is a hyperparameter, which can be adaptively adjusted, potentially representing a future direction for this paper.

Availability

The GitHub repository where the code is hosted is as follows:

- Codes applied on the WVSCM Dataset (Table 8)
- Codes applied on the Fact Evalution Dataset (Table 9)
- Codes applied on the Sentiment Analysis Dataset (Table 10)

References

[1] A. Kittur, J. V. Nickerson, M. Bernstein, E. Gerber, A. Shaw, J. Zimmerman, M. Lease, and J. Horton, "The future of crowd work," in *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, 2013, pp. 1301–1318.

[2] S. Amer-Yahia and S. B. Roy, "Human factors in crowdsourcing," *Proceedings of the VLDB Endowment*, vol. 9, no. 13, pp. 1615–1618, 2016.

[3] T. Buecheler, J. H. Sieg, R. M. Füchslin, and R. Pfeifer, "Crowdsourcing, open innovation and collective intelligence in the scientific method: a research agenda and operational framework," in *The 12th International Conference on the Synthesis and Simulation of Living Systems, Odense, Denmark, 19-23 August 2010.* MIT Press, 2010, pp. 679–686.

[4] B. Li, Y. Cheng, Y. Yuan, Y. Yang, Q. Jin, and G. Wang, "Acta: Autonomy and coordination task assignment in spatial crowdsourcing platforms," *Proceedings of the VLDB Endowment*, vol. 16, no. 5, pp. 1073–1085, 2023.

[5] D. C. Brabham, *Crowdsourcing.* The MIT Press, 2013.

[6] G. Paolacci, J. Chandler, and P. G. Ipeirotis, "Running experiments on amazon mechanical turk," *Judgment and Decision Making*, vol. 5, no. 5, pp. 411–419, 2010.

[7] J. C. Tang, M. Cebrian, N. A. Giacobe, H.-W. Kim, T. Kim, and D. B. Wickert, "Reflecting on the darpa red balloon challenge," *Communications of the ACM*, vol. 54, no. 4, pp. 78–85, 2011.

[8] S. Chatterjee and M. Bhattacharyya, "Judgment analysis of crowdsourced opinions using biclustering," *Information Sciences*, vol. 375, pp. 138–154, 2017.

[9] P. G. Ipeirotis, "Analyzing the amazon mechanical turk marketplace," *XRDS: Crossroads, The ACM magazine for students*, vol. 17, no. 2, pp. 16–21, 2010.

[10] K. J. Boudreau, N. Lacetera, and K. R. Lakhani, "Incentives and problem uncertainty in innovation contests: An empirical analysis," *Management science*, vol. 57, no. 5, pp. 843–863, 2011.

[11] T. X. Liu, J. Yang, L. A. Adamic, and Y. Chen, "Crowdsourcing with all-pay auctions: A field experiment on taskcn," *Management Science*, vol. 60, no. 8, pp. 2020–2037, 2014.

[12] K. J. Boudreau and K. R. Lakhani, "Using the crowd as an innovation partner." *Harvard business review*, vol. 91, no. 4, pp. 60–9, 2013.

[13] R. Snow, B. O'connor, D. Jurafsky, and A. Y. Ng, "Cheap and fast–but is it good? evaluating non-expert annotations for natural language tasks," in *Proceedings of the 2008 conference on empirical methods in natural language processing*, 2008, pp. 254–263.

[14] A. Sorokin and D. Forsyth, "Utility data annotation with amazon mechanical turk," in *2008 IEEE computer society conference on computer vision and pattern recognition workshops*. IEEE, 2008, pp. 1–8.

[15] S. Chatterjee, A. Mukhopadhyay, and M. Bhattacharyya, "A review of judgment analysis algorithms for crowdsourced opinions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 7, pp. 1234–1248, 2019.

[16] S. Muthukrishnan *et al.*, "Data streams: Algorithms and applications," *Foundations and Trends® in Theoretical Computer Science*, vol. 1, no. 2, pp. 117–236, 2005.

[17] R. Rubinfeld and A. Shapira, "Sublinear time algorithms," *SIAM Journal on Discrete Mathematics*, vol. 25, no. 4, pp. 1562–1588, 2011.

[18] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SIAM Journal on Computing*, vol. 31, no. 6, pp. 1794–1813, 2002.

[19] J. Whitehill, T.-f. Wu, J. Bergsma, J. Movellan, and P. Ruvolo, "Whose vote should count more: Optimal integration of labels from labelers of unknown expertise," *Advances in Neural Information Processing Systems*, vol. 22, 2009.

[20] L. R. Varshney, S. Agarwal, Y.-M. Chee, R. R. Sindhgatta, D. V. Oppenheim, J. Lee, and K. Ratakonda, "Cognitive coordination of global service delivery," *arXiv preprint arXiv:1406.0215*, 2014.