Introduction

Problem: The project aims to solve the problem of determining whether an animal is in a dangerous condition based on observed symptoms. The integration of this functionality into the PetDesk app will assist pet owners in making informed decisions about seeking immediate medical attention for their pets.

Impact: Successfully solving this problem will significantly enhance pet healthcare management, enabling pet owners to quickly identify health risks and seek timely veterinary intervention, thereby improving the overall well-being and safety of pets.

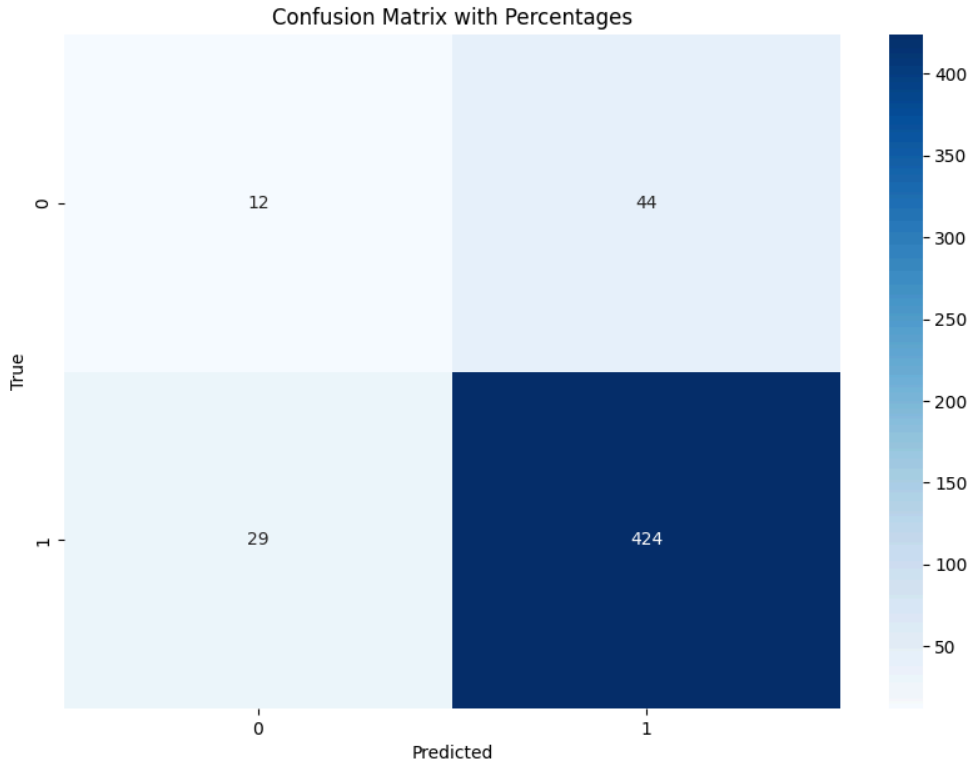
Formulation

We can describe it as a classification type data mining task. The input is AnimalName, symptoms1, symptoms2, symptoms3, symptoms4 symptoms5, and Dangerous. For these 7 pieces of data, the prefetch output should be whether the pet is currently in a crisis state (Dangerous: Yes/No)

For example:

Input: Dog, Fever, Diarrhea, Vomiting, Weight loss, Dehydration, Yes

Output:



Datasets

The dataset, titled "Animal Condition Classification Dataset," was obtained from the following source: [Kaggle - Animal Condition Dataset]. It consists of:

- Instances: 871 records
- Features:
    - AnimalName: The type of animal, with categories such as 'Buffaloes', 'Sheep', and many other unspecified categories grouped as 'Other'.
    - symptoms1 to symptoms5: Five symptom attributes that describe the condition of the animals, with examples including 'Fever', 'Diarrhea', 'Coughing', 'Weight loss', and 'Pains'.
    - Target Variable: A binary target variable indicating whether the condition is dangerous, 'Dangerous' (Yes/No)
- Data Preprocessing

- The symptoms are encoded and converted from string format to int format to better match the needs of the algorithm model.
- Some incomplete data were removed because these data would interfere with the model.
- The data is divided into a training set and a test set to test the stability of the algorithm.

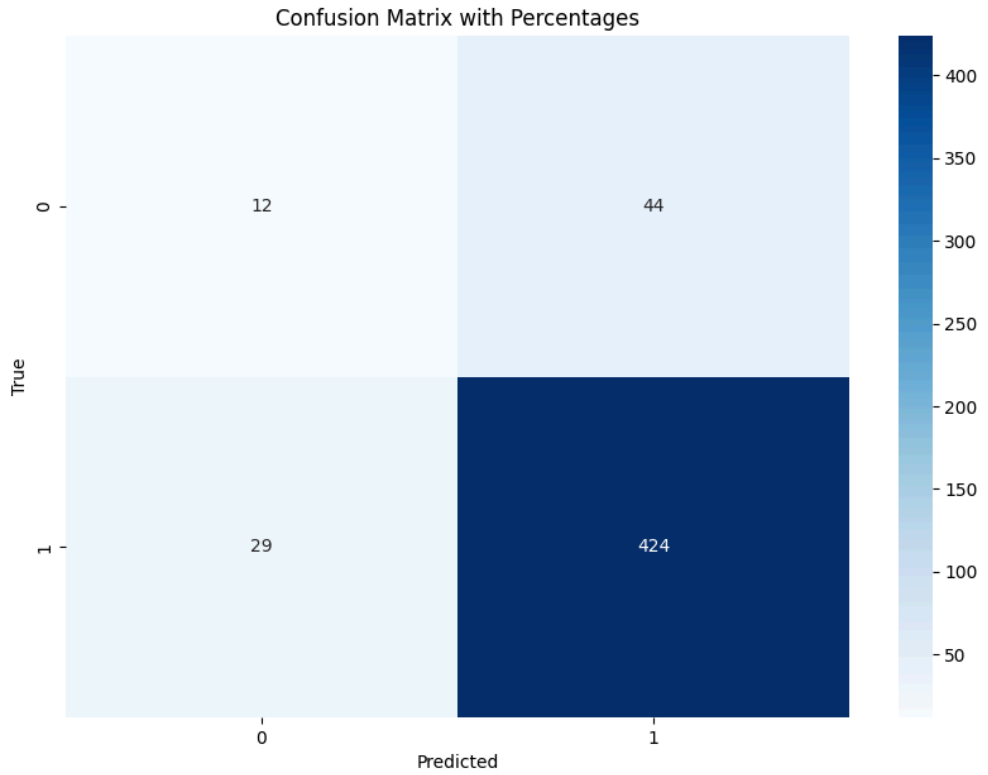
AnimalName aniname		symptoms1 symptoms		symptoms2 symptoms2		symptoms3 symp3	
Buffaloes	15%	Fever	30%	Diarrhea	14%	Coughing	11%
Sheep	13%	Fetopelvic dispropo...	2%	Difficulty in breathing	3%	Vomiting	7%
Other (632)	73%	Other (593)	68%	Other (726)	83%	Other (717)	82%

symptoms4 symp4		symptoms5 symp5		Dangerous danger	
Weight loss	13%	Pains	11%	true 849 97%	
Death	6%	Pain	8%	false 20 2%	
Other (703)	81%	Other (704)	81%	[null] 2 0%	

Algorithm

Decision tree

Experiments



The output result graph is divided into predicted data and actual data. It is a confusion matrix.

- **True Negatives (TN): 12**
- **False Positives (FP): 44**
- **False Negatives (FN): 29**
- **True Positives (TP): 424**
- **Accuracy: 85.66%**
- **Precision: 90.6%**
- **Recall: 93.6%**
- **F-measure(F): 92.07%**

Identifying true positive results and being generally accurate, these results are meaningful.

Challenges

- Challenge: Data Quality – Inconsistencies in symptom descriptions due to manual data entry posed a significant challenge.
    - Solution: Rigorous data cleaning and validation procedures were implemented to improve data quality.
- Challenge: Class Imbalance – Skewed distribution of the 'Dangerous' classification required careful handling to prevent model bias.
    - Solution: Advanced balancing techniques were utilized to mitigate class imbalance issues.
- Challenge: Feature Engineering – Deriving meaningful features from the symptoms was crucial and challenging.
    - Solution: Expert domain knowledge was applied to create additional features that capture the essence of the symptoms more effectively.

```
#include "../include/simulator.h"
#include "stdbool.h"
#include "stdio.h"
#include "string.h"

#define A 0
#define B 1
#define MAX_BUFF 1000
#define MSG_SIZE 20

bool IsWaitingForAck;
int SeqNumB, LastAckNum, CurrentSeq, NextSeq, WinBase, WinSize;
struct pkt Buffer[MAX_BUFF];


int Checksum(struct pkt packet) {
 int sum = packet.seqnum + packet.acknum;
 for (int i = 0; i < MSG_SIZE && packet.payload[i] != '\0'; ++i) {
   sum += packet.payload[i];
```

```
 }
 return sum;
}

void A_output(struct msg message) {
 if (CurrentSeq < MAX_BUFF) {
   struct pkt packet;
   packet.seqnum = CurrentSeq;
   memcpy(packet.payload, message.data, MSG_SIZE);
   packet.checksum = Checksum(packet);

   Buffer[CurrentSeq % WinSize] = packet;
   if (NextSeq < WinBase + WinSize) {
     tolayer3(A, packet);
     if (WinBase == NextSeq) starttimer(A, 20.0);
     NextSeq++;
   }
   CurrentSeq++;
 }
}

void A_input(struct pkt packet) {
 if (packet.acknum >= WinBase && packet.checksum ==
Checksum(packet)) {
   WinBase = packet.acknum + 1;
   stoptimer(A);
   if (NextSeq < CurrentSeq && NextSeq < WinBase + WinSize) {
     tolayer3(A, Buffer[NextSeq % WinSize]);
     NextSeq++;
     starttimer(A, MSG_SIZE);
   }
 }
}

void A_timerinterrupt() {
 starttimer(A, MSG_SIZE);
 for (int i = WinBase; i < NextSeq; ++i) {
   tolayer3(A, Buffer[i % WinSize]);
 }
}
```

```
void A_init() {
 CurrentSeq = NextSeq = WinBase = 0;
 WinSize = getwinsize();
}

void B_input(struct pkt packet) {
 struct pkt ack_pkt;
 ack_pkt.acknum = packet.seqnum;
 ack_pkt.checksum = Checksum(ack_pkt);

 if (packet.checksum == Checksum(packet) && packet.seqnum ==
SeqNumB) {
   tolayer5(B, packet.payload);
   SeqNumB++;
 } else {
   ack_pkt.acknum = LastAckNum;
 }
 tolayer3(B, ack_pkt);
 LastAckNum = packet.seqnum;
}

void B_init() {
 SeqNumB = LastAckNum = 0;
}
```

```c
#include "../include/simulator.h"
#include "stdbool.h"
#include "stdio.h"
#include "string.h"

#define A 0
#define B 1
#define BufferSize 1000
#define FileSize 20

int B_ENTRY, Next_ENTRY, window_size, SendB;
struct pkt BufferPacket[BufferSize];

int calculateChecksum(struct pkt packet)
{
   int checksum = packet.seqnum + packet.acknum;
   for (int i = 0; i < FileSize && packet.payload[i] != '\0'; i++) {
       checksum += packet.payload[i];
   }
   return checksum;
}

/* ********************************************************************
ALTERNATING BIT AND GO-BACK-N NETWORK EMULATOR: VERSION 1.1  J.F.Kurose

  This code should be used for PA2, unidirectional data transfer
  protocols (from A to B). Network properties:
  - one way network delay averages five time units (longer if there
    are other messages in the channel for GBN), but can be larger
  - packets can be corrupted (either the header or the data portion)
    or lost, according to user-defined probabilities
  - packets will be delivered in the order in which they were sent
    (although some can be lost).
**********************************************************************/

/********* STUDENTS WRITE THE NEXT SEVEN ROUTINES *********/

/* called from layer 5, passed the data to be sent to other side */
void sendPacket(int seqNum, char *data) {
   struct pkt newPacket;
```

```
    newPacket.seqnum = seqNum;
    memcpy(newPacket.payload, data, FileSize);
    newPacket.checksum = calculateChecksum(newPacket);
    tolayer3(A, newPacket);
    BufferPacket[seqNum % window_size] = newPacket; // Store the packet
}

void A_output(struct msg message) {
    if (Next_ENTRY < SendB + window_size) {
        sendPacket(Next_ENTRY, message.data); //
        if (Next_ENTRY == SendB) {
            starttimer(A, 20.0); // Start timer for the first packet in the
window
        }

        Next_ENTRY++;
    }
}




/* called from layer 3, when a packet arrives for layer 4 */
void A_input(struct pkt packet){
    if (packet.checksum == calculateChecksum(packet)) {
        if (packet.acknum >= SendB) {
            SendB = packet.acknum + 1;
            if (SendB == Next_ENTRY) {
                stoptimer(A);
            } else {
                // restart the timer
                starttimer(A, 20.0);
            }
        }
    }
}

/* called when A's timer goes off */
void A_timerinterrupt(){
    starttimer(A, 20.0); // Restart the timer
```

```
    for (int i = SendB; i < Next_ENTRY; i++) {
        // Retransmit all packets in the window
        tolayer3(A, BufferPacket[i % window_size]);
    }
}

/* the following routine will be called once (only) before any other */
/* entity A routines are called. You can use it to do any initialization
*/
void A_init()
{
    SendB = Next_ENTRY = 0;
    window_size = getwinsize(); // Set the window size
}

/* Note that with simplex transfer from a-to-B, there is no B_output() */

/* called from layer 3, when a packet arrives for layer 4 at B*/
void B_input(struct pkt packet)
{
    struct pkt ack_packet;
    ack_packet.acknum = packet.seqnum;
    ack_packet.checksum = calculateChecksum(ack_packet);

    if (packet.checksum == calculateChecksum(packet) && packet.seqnum ==
B_ENTRY) {
        tolayer5(B, packet.payload);
        B_ENTRY++;
    }
    // Send an ACK for the received packet
    tolayer3(B, ack_packet);
}

/* the following routine will be called once (only) before any other */
/* entity B routines are called. You can use it to do any initialization
*/
void B_init()
{
    B_ENTRY = 0; // Expected sequence number at B-side
}
```