

Classification and Analysis of Multiple Cattle Unitary Behaviors on Machine Learning Methods (Predictive Analysis)

Need:

Traditionally, farmers are unable to pay enough attention to individual livestock. An increasing number of sensors are being used to monitor animal behavior, early disease detection, and evaluation of animal welfare.

we can use machine learning algorithms to identify multiple unitary behaviors of dairy cattle recorded by motion sensors. We also investigated the effect of time window on the performance of unitary behaviors classification and discussed the necessity of movement analysis.

Low-cost sensors(cow sensor/accelerometer) provide remote monitoring of animal behaviors to help producers comprehensively and accurately identify the health status of individual livestock in real-time.

Sensor system:

Accelerometer Data (x, y, z) or IMU is used.

Data Collection:

1. Video Data Collection: - Set up video recording equipment in the cattle barn. Ensure they cover a substantial portion of the barn to capture cattle behavior.

2. Video Recording Setup: - Set up the cameras to record video continuously during your data collection period (example: 24hrs). Ensure proper lighting and camera positioning to obtain clear video footage.

4. Unitary Behaviors/ Classes: - Define the behaviors you want to observe. Based on your description, you have already defined six unitary behaviors: feeding, standing, lying, ruminating-standing, ruminating-lying, and walking.

[[Capture.PNG]]

5. Behavioral Annotation: - Designate one or more trained observers who will watch the recorded videos and annotate cattle behaviors based on your predefined definitions. - Consider using video annotation software or creating a custom annotation tool in Python to facilitate the labeling process. This tool can allow observers to label behaviors while watching the video. - Your trained observers should watch the videos and label each time segment with one of the predefined behaviors, such as “feeding,” “standing,” and so on. - Segment the video data into behavior segments. Based on your description, these segments should be at least 30 seconds in duration. - Example: The dataset included 400 individual behavior segments (each segment more than 30 s), with a total of approximately 78.6 h. -

[[Capture2.PNG]]

raw program:

```

import csv

# Define the list of predefined behaviors
behaviors = ["feeding", "standing", "lying", "ruminating-standing", "ruminati
ng-lying", "walking"]

# Create a list to store annotations
annotations = []

# Function to annotate a video segment
def annotate_segment(start_time, end_time, behavior):
    print(f"Annotating segment from {start_time} to {end_time} as '{behavior}'")

    # Add the annotation to the list
    annotations.append([start_time, end_time, behavior])

# Function to calculate the total duration of the CSV file
def calculate_total_duration(input_csv):
    with open(input_csv, 'r') as csvfile:
        reader = csv.reader(csvfile)
        data = list(reader)

    total_duration = float(data[-1][0]) # Assuming time duration is in the l
ast row
    return total_duration

# Function to divide the CSV file into segments
def divide_csv_into_segments(input_csv, num_segments):
    total_duration = calculate_total_duration(input_csv)
    segment_duration = total_duration / num_segments

    with open(input_csv, 'r') as csvfile:
        reader = csv.reader(csvfile)
        data = list(reader)

    current_time = 0.0

    for i in range(num_segments):
        start_time = current_time
        end_time = current_time + segment_duration

        yield data, start_time, end_time

        current_time = end_time

# Main function to annotate video segments
def annotate_video_segments(input_csv, num_segments):
    for segment_data, start_time, end_time in divide_csv_into_segments(input_

```

```

csv, num_segments):
    print(f"Segment from {start_time} to {end_time}:\n")

    # Display available behaviors
    print("Available behaviors:")
    for i, behavior in enumerate(behaviors):
        print(f"{i + 1}. {behavior}")

    # Prompt the user to select a behavior
    while True:
        try:
            choice = int(input("Enter the number corresponding to the beh
avior: "))
            if 1 <= choice <= len(behaviors):
                selected_behavior = behaviors[choice - 1]
                break
            else:
                print("Invalid choice. Please enter a valid number.")
        except ValueError:
            print("Invalid input. Please enter a number.")

    annotate_segment(start_time, end_time, selected_behavior)

# Get user input for CSV file and number of segments
input_csv = input("Enter the path to the input CSV file: ")
num_segments = int(input("Enter the number of segments: "))

# Annotate video segments
annotate_video_segments(input_csv, num_segments)

# Write the annotations to a CSV file
output_csv = "labeled_annotations.csv"
with open(output_csv, "w", newline="") as csv_file:
    csv_writer = csv.writer(csv_file)
    csv_writer.writerow(["Start Time", "End Time", "Behavior"])
    csv_writer.writerows(annotations)


print(f"Annotations saved to '{output_csv}'.")

```

1. The program is designed to annotate video segments based on a predefined list of behaviors.
2. It takes an input CSV file containing time duration and accelerometer data (x, y, z values) as the source data for annotation.
3. The list of predefined behaviors includes “feeding,” “standing,” “lying,” “ruminating-standing,” “ruminating-lying,” and “walking.”
4. The user can specify the number of segments into which the video data should be divided.
5. The program calculates the total duration of the video from the input CSV file.

6. It then divides the data into equal segments based on the user's input, ensuring the segments cover the entire duration of the video.
7. For each segment, the program displays the available behaviors and prompts the user to choose a behavior label for that segment.
8. The user can select the appropriate behavior by entering the corresponding number from the displayed list.
9. The program annotates each segment with the chosen behavior and records the start and end times of the segment.
10. The annotated data is saved to a CSV file, providing labeled annotations for each video segment.

6. Unitary Behavior Classification and Analysis

The overall processing flow for behaviors classification and analysis is provided in Figure 2. The framework can be divided into two stages: (1) long segments that were continuous and undivided during unitary behaviors and (2) the movements of the feeding segments correctly classified in the first step. Feature extraction, model construction, and analysis were all implemented using python 3.7. 

7. data pre-processing:

1. **Data Sources:**
 - You have two primary data sources: motion sensor data and video data, both collected from cattle.
2. **Sliding Time Window:**
 - A sliding time window of 5 seconds is applied to the time series data from motion sensors. This window slides with a 50% overlap, allowing for continuous analysis of the data.
3. **Behavior Labeling:**
 - Each time window contains a behavior label. This label is crucial for supervised learning, as it represents the ground truth of the behavior observed during that window.
4. **Window Sizes:**
 - You have defined four different window sizes. The choice of window size can affect the granularity of your analysis. Larger windows may capture longer-term behaviors, while smaller windows may capture finer details.
5. **Feature Extraction:**
 - Time-Domain Features:
 - Features such as maximum, minimum, mean, variance, standard deviation, peak-to-peak distance, and mode are computed for each time window.
 - These statistics provide valuable insights into the statistical properties of the sensor data during each window.
 - Frequency-Domain Features:

- Features in the frequency domain are computed after applying the Fast Fourier Transform (FFT) to the time series data.
- These features include direct current, power spectral density, spectral entropy, and spectral energy. They capture frequency components in the data.
- Additional Features:
 - Two domain-specific features, overall dynamic body acceleration (ODBA) and movement variation (MV), are also calculated.
 - These features are based on previous research and provide insights into cattle movements.

6. Normalization:

- All thirteen features extracted from each time window are normalized to the 0-1 range. Normalization ensures that features with different scales do not dominate the analysis and model training.

The processed data will be used for further analysis and classification tasks. This involves training machine learning models to classify behaviors and movements based on the extracted features and behavior labels. The choice of window size and the selection of relevant features should be made carefully to ensure that your models capture meaningful patterns and behaviors.

The entire data preprocessing pipeline should be implemented using Python, and libraries such as NumPy, pandas, and scikit-learn can be helpful for data manipulation and feature extraction. Additionally, you may use the features extracted during data preprocessing to train and evaluate machine learning models as part of your classification and analysis process.

raw program: Please note that this is a simplified example and does not include specific libraries for FFT, ODBA, MV, or normalization. You may need to use libraries like NumPy, pandas, SciPy, and scikit-learn to implement these steps effectively.

```
import pandas as pd

# Load the labeled data (CSV file with timestamps and behavior labels)
labeled_data = pd.read_csv("labeled_data.csv")

# Define the sliding time window parameters
window_size = 5 # 5 seconds
overlap = 0.5 # 50% overlap

# Initialize lists to store extracted features and behavior labels
features = []
labels = []

# Calculate the overlap duration (half of the window size)
overlap_duration = window_size * overlap
```

```

# Iterate over the Labeled data
for index, row in labeled_data.iterrows():
    timestamp = row['Timestamp']
    behavior_label = row['Behavior']

    # Determine the start and end of the time window
    window_start = timestamp - window_size / 2
    window_end = timestamp + window_size / 2

    # Extract data points within the time window
    data_within_window = your_data_extraction_function(window_start, window_e
nd)

    # Calculate time-domain and frequency-domain features from data_within_wi
ndow
    time_features = calculate_time_domain_features(data_within_window)
    freq_features = calculate_frequency_domain_features(data_within_window)

    # Combine all features
    all_features = time_features + freq_features

    # Normalize the features to the 0-1 range (implement this function)
    normalized_features = normalize_features(all_features)

    # Append the features and behavior Label
    features.append(normalized_features)
    labels.append(behavior_label)


# Create a DataFrame to store the preprocessed data
preprocessed_data = pd.DataFrame(features, columns=['Feature1', 'Feature2', .
..])
preprocessed_data['Behavior'] = labels

# Save the preprocessed data to a CSV file
preprocessed_data.to_csv("preprocessed_data.csv", index=False)

```

In this template:

- Load your labeled data (timestamps and behavior labels) from a CSV file.
- Define the sliding time window parameters.
- Iterate over the labeled data, extract data points within each time window.
- Calculate time-domain and frequency-domain features for the data within each window.
- Normalize the extracted features to the 0-1 range.
- Create a new DataFrame to store the preprocessed data, including both features and behavior labels.
- Save the preprocessed data to a new CSV file for further use.

You would need to replace the placeholders such as `your_data_extraction_function`, `calculate_time_domain_features`, and `calculate_frequency_domain_features` with your specific data extraction and feature calculation functions based on your dataset and analysis needs. 

8. Machine Learning Models:

1. K-Nearest Neighbors (KNN):

- KNN is a simple and effective classification algorithm.
- It classifies samples by finding the K nearest neighbors from the training data.
- Distance metrics like Euclidean or Manhattan distance are used to measure similarity.
- It's important to normalize features to ensure they have the same scale.
- The optimal value of K is often determined through cross-validation.

2. Random Forest (RF):

- RF is an ensemble learning method that combines multiple decision trees.
- It involves bootstrapping (randomly selecting samples with replacement) to train individual decision trees.
- Each decision tree votes on the class, and the majority vote becomes the final prediction.
- RF is known for its robustness and resistance to overfitting.

3. Extreme Boosting Algorithm (XGBoost):

- XGBoost is a boosting ensemble algorithm that builds decision trees sequentially.
- It fits a new tree to the residuals of the previous trees, gradually improving prediction.
- A penalty term is added to the loss function to prevent overfitting.
- XGBoost is known for its efficiency and high performance in various classification tasks.

To proceed further:

1. Evaluate the performance of each model using appropriate metrics (e.g., accuracy, precision, recall, F1 score).
2. Compare the performance of these models to select the most suitable one for your specific cattle unitary behavior classification task.
3. Fine-tune the selected model if necessary by adjusting hyperparameters.
4. Implement the chosen model on new, unseen data for real-world applications.