Ahmedabad University

CSE623 - Machine Learning Theory and Practice
Section - 1

Weekly Report 4
Identify Hard stop and momentary stop using the vehicle trajectory dataset

---

Team Name: **The Overfitters**

Name	Enrolment Number
Jinil Savaj	AU2240159
Jay Raval	AU2240151
Meet Suthar	AU2240198
Karan Prajapati	AU2240161
Vishesh Bhatia	AU2240027

Previous Approach

Until the last report, we used a threshold-based method to label our dataset. We manually determined threshold values for velocity and acceleration by analyzing plotted graphs. Using these thresholds, we classified vehicle states into three categories:

- **Hard Stop (2)**
- **Momentary Stop (1)**
- **Moving (0)**

The dataset was then trained on a Random Forest model, which provided reasonable classification accuracy. However, this method relied heavily on manual threshold selection, which might not generalize well to new data.

New Methods for Labeling the Dataset

To improve labeling and possibly achieve better accuracy, we explored two alternative approaches: **Convexity-Based Stop Detection** and **K-Means Clustering for Labeling.**

Method 1: Convexity-Based Stop Detection

Instead of using arbitrary thresholds, this method leverages the shape (convexity) of velocity and acceleration curves to detect stops. It follows these steps:

1. **Smooth the velocity and acceleration** using a rolling mean to reduce noise.
2. **Identify local minima** in velocity, which indicate potential stops.
3. **Analyze convexity properties:**
    - If the velocity curve is convex around the minimum, it suggests a momentary stop.

- If velocity remains nearly flat with little acceleration variance, it's classified as a hard stop.

Key Code Snippet:

```
import numpy as np
from scipy.signal import find_peaks

def label_stops(df, min_stop_width=3):
    vel = df['velocity'].rolling(5, 
center=True).mean().fillna(df['velocity'])
    accel = df['acceleration'].rolling(5, 
center=True).mean().fillna(df['acceleration'])

    minima, _ = find_peaks(-vel, width=min_stop_width)
    labels = np.zeros(len(df))  # Default: moving

    for i in minima:
        left = max(0, i-5)
        right = min(len(df)-1, i+5)
        convex_score = (vel[left] + vel[right]) / 2 - vel[i]
        accel_changes = accel[left:i].mean() - accel[i:right].mean()

        if (np.abs(vel[left:right].std()) < 0.1 and 
np.abs(accel[left:right].std()) < 0.05):
            labels[left:right] = 2  # Hard stop
        elif convex_score > 0 and np.abs(accel_changes) > 0.1:
            labels[i] = 1  # Momentary stop

    return labels

df['stop_type'] = label_stops(df)
```

Method 2: K-Means Clustering for Labeling

This method removes the need for manual labeling by using **unsupervised learning** to identify patterns in vehicle behavior. It follows these steps:

1. **Feature Engineering:**
    - Raw velocity and acceleration.
    - Rolling mean and standard deviation.
    - Jerk (rate of change of acceleration).
2. **Feature Scaling:** Standardize the features for better clustering.
3. **Apply K-Means Clustering:** Assigns each data point to one of three clusters.
4. **Interpret Cluster Labels:** Identify which cluster represents moving, momentary stops, and hard stops based on their average velocity and acceleration values.

Key Code Snippet:

```
from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler



def label_with_kmeans(df, n_clusters=3):

    features = pd.DataFrame({

        'velocity': df['velocity'],

        'acceleration': df['acceleration'],

        'vel_rolling_mean': df['velocity'].rolling(5,
center=True).mean().fillna(df['velocity']),

        'accel_rolling_std': df['acceleration'].rolling(5,
center=True).std().fillna(0),

        'jerk': df['acceleration'].diff().abs()

    })
    scaler = StandardScaler()

    scaled_features = scaler.fit_transform(features.dropna())

    kmeans = KMeans(n_clusters=n_clusters, random_state=42)

    labels = kmeans.fit_predict(scaled_features)

    final_labels = np.full(len(df), -1)

    valid_indices = features.dropna().index

    final_labels[valid_indices] = labels

    return final_labels

df['cluster_label'] = label_with_kmeans(df)
```

Interpreting Clusters:

```
def interpret_clusters(df):

    cluster_stats = df.groupby('cluster_label')[['velocity',
'acceleration']].mean()

    moving_cluster = cluster_stats['velocity'].idxmax()

    hard_stop_cluster = cluster_stats['velocity'].idxmin()

    momentary_cluster = [x for x in cluster_stats.index if x not in
[moving_cluster, hard_stop_cluster]][0]



    label_map = {

        moving_cluster: 0,     # Moving

        momentary_cluster: 1, # Momentary stop

        hard_stop_cluster: 2  # Hard stop
    }

    return df['cluster_label'].map(label_map).fillna(0)

df['stop_type'] = interpret_clusters(df)
```

Next Steps:

- Evaluate both new methods by training the Random Forest model and comparing accuracy with the threshold-based approach.
- Check whether unsupervised clustering (K-Means) or convexity-based labeling results in better classification.
- Fine-tune hyperparameters like rolling window size and cluster count if needed.

These new methods aim to improve labeling automation, reduce manual intervention, and potentially boost model performance. The evaluation results will guide which labeling method to finalize for further development.