OPENCAMPUS.sh

PREDICTING INDIVIDUAL DRIVING HABITS

Project By:

Maciej Lipski (ID: 1198270)

Sai Mohit (ID: 946644)

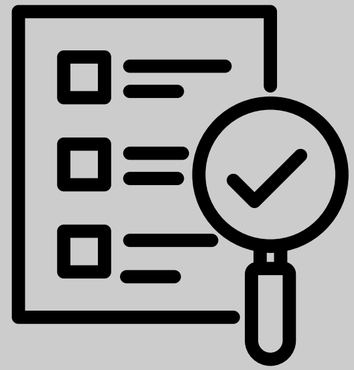
Clifford Daniel (ID: 942820)

16.01.2025

INTRODUCTION

This project aims to predict cyclists' speed using GPX files collected from Slovenian professional cyclists. Initially, the objective was to estimate time based on these files. However, due to insufficient data for accurate time prediction, the focus shifted to speed prediction after building and evaluating the base model.

Inputs:
Elevation, Slope, Angle, Distance, Cumulative Slope
Output:
Speed, Estimated Time



LITERATURE REVIEW

Source 1

Bicycle Data-Driven Application Framework: A Dutch Case Study on Machine Learning-Based Bicycle Delay Estimation at Signalized Intersections Using Nationwide Sparse GPS Data

Source 2

The Development of a Predictive Hiking Travel Time Model Accounting for Terrain Variations

Source 3

Bike Share Travel Time Modeling: San Francisco Bay Area Case Study

DATASET CHARACTERISTICS

- The dataset was taken from a technical report on nine professional cyclists, with data captured from Garmin or Strava in GPX or TCX file formats.

- **Attributes Captured**
  - GPS Location data
  - Elevation
  - Duration of activities
  - Distance Covered
  - Average and maximal heart rate

- **Tools and Libraries**
  - Used GPX viewer to look at the files
  - Used gpxpy and tcxreader

```
Folder 'Rider1' contains 751 .gpx files and 0 .tcx files.
Folder 'Rider2' contains 389 .gpx files and 0 .tcx files.
Folder 'Rider3' contains 661 .gpx files and 0 .tcx files.
Folder 'Rider4' contains 810 .gpx files and 0 .tcx files.
Folder 'Rider5' contains 636 .gpx files and 0 .tcx files.
Folder 'Rider6' contains 63 .gpx files and 0 .tcx files.
Folder 'Rider7' contains 328 .gpx files and 0 .tcx files.
Folder 'Rider8' contains 0 .gpx files and 170 .tcx files.
Folder 'Rider9' contains 0 .gpx files and 585 .tcx files.
```

```
Number of samples: 5994
Number of features: 4
Example data:
                          time   latitude  longitude  elevation
0 2013-07-13 06:41:38+00:00  46.363649  14.112036      511.5
1 2013-07-13 06:41:39+00:00  46.363691  14.112026      511.0
2 2013-07-13 06:41:41+00:00  46.363778  14.112031      510.1
3 2013-07-13 06:41:43+00:00  46.363835  14.112023      509.7
4 2013-07-13 06:41:45+00:00  46.363888  14.112007      509.4
```

STRAVA GARMIN

DATA CONVERSION AND PRE-PROCESSING

- Delete files recorded outside Europe and Remove Corrupted files
- Calculate slope, angle, speed
- Convert timestamp to absolute time
- Combine multiple files into one as Rider's file

```python
def preprocess_rider_data(input_dir, output_dir):
    for filename in os.listdir(input_dir):
        if filename.endswith(".xlsx"):
            filepath = os.path.join(input_dir, filename)
            try:
                df = pd.read_excel(filepath)

                # Step 1: If file is f97.xlsx, delete the first 11 rows
                if filename == "f97.xlsx":
                    df = df.iloc[11:].reset_index(drop=True)

                # Step 2: Delete the file if max_longitude < 10 or max_latitude < 40
                if df['Longitude'].max() < 10 or df['Latitude'].max() < 40:
                    print(f"File {filename} deleted due to coordinates.")
                    os.remove(filepath)
                    continue  # Skip the rest of the loop for this file

                # Step 3: Convert timestamp to seconds
                df['Time'] = pd.to_datetime(df['Time'])
                df['Time'] = (df['Time'] - df['Time'].iloc[0]).dt.total_seconds()

                # Step 4: Calculate distance since the beginning
                distances = []
                cumulative_distance = 0
                for i in range(len(df)):
                    if i > 0:
                        previous_coords = (df['Latitude'].iloc[i-1], df['Longitude'].iloc[i-1])
                        current_coords = (df['Latitude'].iloc[i], df['Longitude'].iloc[i])
                        distance = geodesic(previous_coords, current_coords).meters
                        cumulative_distance += distance
                    else:
                        distance = 0
                    distances.append(cumulative_distance)
                df['Distance'] = distances

                # Step 5: Save the preprocessed file
                output_filepath = os.path.join(output_dir, filename)
                df.to_excel(output_filepath, index=False)
                print(f"Processed and saved: {filename}")
            except Exception as e:
                print(f"Error processing {filename}: {e}")
```

BASELINE MODEL

- Linear Regression was chosen due to simplicity, speed of calculations and conceptial fit to the task.
- Linear Regression results (percentage difference on the test set):
  - Rider 1: 29.59
  - Rider 3: 15.36
  - Rider 7: 26.02

	file	last_time	last_time_pred	percentage_diff
3	lr_f673.csv	126	94.968318	-24.628319
4	lr_f562.csv	1283	593.353649	-53.752638
6	lr_f123.csv	3791	3709.756866	-2.143053
1	lr_f660.csv	5839	6407.278490	9.732463
8	lr_f196.csv	6416	6655.971591	3.740206
2	lr_f80.csv	9183	3911.889730	-57.400743
0	lr_f114.csv	9630	7157.584003	-25.674102
5	lr_f139.csv	11311	11480.384531	1.497520
9	lr_f593.csv	11581	4104.998230	-64.554026
7	lr_f234.csv	19531	9217.082911	-52.807931

Percentage difference in LR model: 29.59310014727181

	file	last_time	last_time_pred	percentage_diff
8	lr_f114.csv	5553	5600.173762	0.849518
7	lr_f144.csv	6864	6336.960079	-7.678321
4	lr_f234.csv	4294	3540.409084	-17.549858
5	lr_f24.csv	8877	9524.708456	7.296479
0	lr_f437.csv	1138	766.798228	-32.618785
1	lr_f47.csv	5492	5804.706142	5.693848
6	lr_f537.csv	4311	4608.122235	6.892188
9	lr_f559.csv	1412	980.967598	-30.526374
2	lr_f566.csv	2290	1332.727136	-41.802308
3	lr_f647.csv	5229	5369.212654	2.681443

Absolute percentage difference in LR model: 15.358912331780052

	file	last_time	last_time_pred	percentage_diff
2	lr_f102.csv	5436	7493.707860	37.853345
9	lr_f159.csv	2380	2022.947238	-15.002217
6	lr_f160.csv	10727	12113.556890	12.925859
1	lr_f164.csv	2035	2153.199542	5.808331
5	lr_f205.csv	9728	10947.301412	12.533937
8	lr_f239.csv	9058	9654.154531	6.581525
3	lr_f269.csv	6972	9275.629140	33.041152
0	lr_f39.csv	8448	10552.621310	24.912658
7	lr_f5.csv	952	51.383827	-94.602539
4	lr_f87.csv	10328	12079.455753	16.958324

Absolute percentage difference in LR model: 26.02198883271553

MODEL SUMMARY

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	896
dense_1 (Dense)	(None, 256)	33,024
dropout (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32,896
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 1)	129

Total params: 66,945 (261.50 KB)

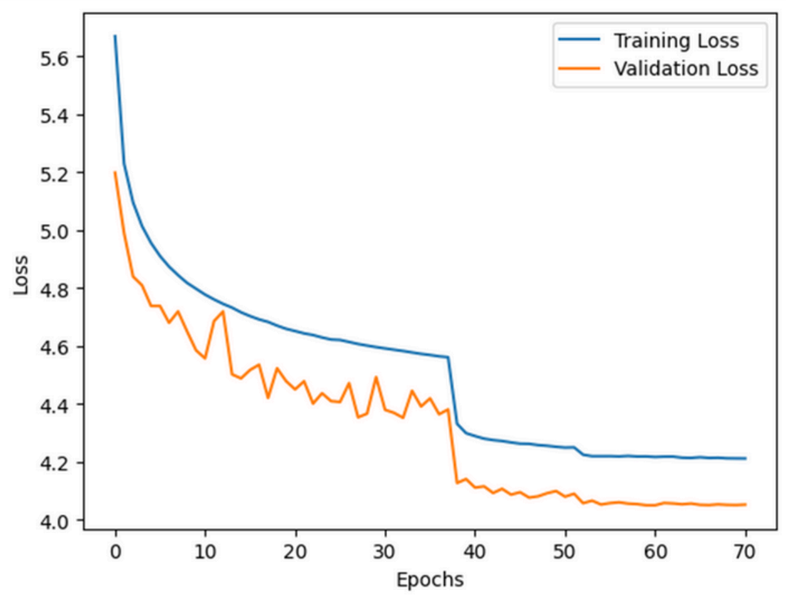
Trainable params: 66,945 (261.50 KB)

Non-trainable params: 0 (0.00 B)

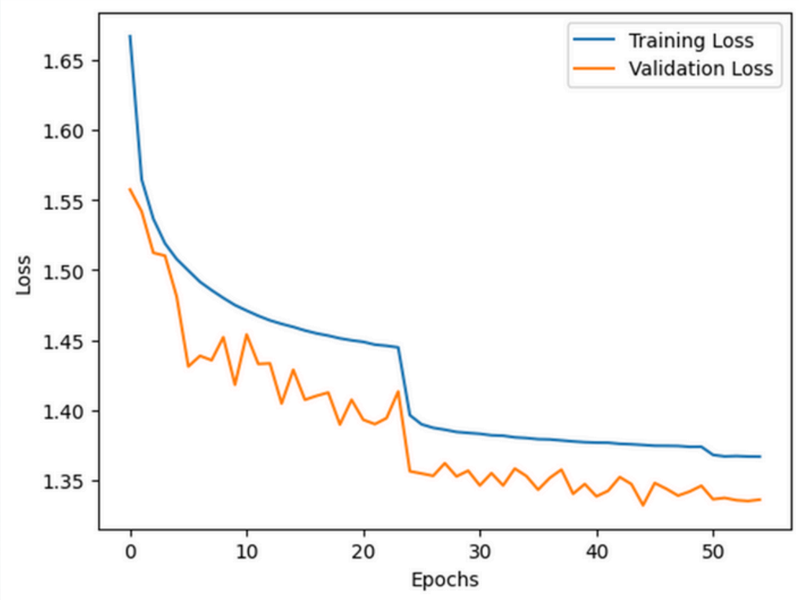
EVALUATION METRIC

NO SIGNIFICANT DIFFERENCE BETWEEN METRICS
MSE CHOSEN TO PENALIZE LARGER ERRORS

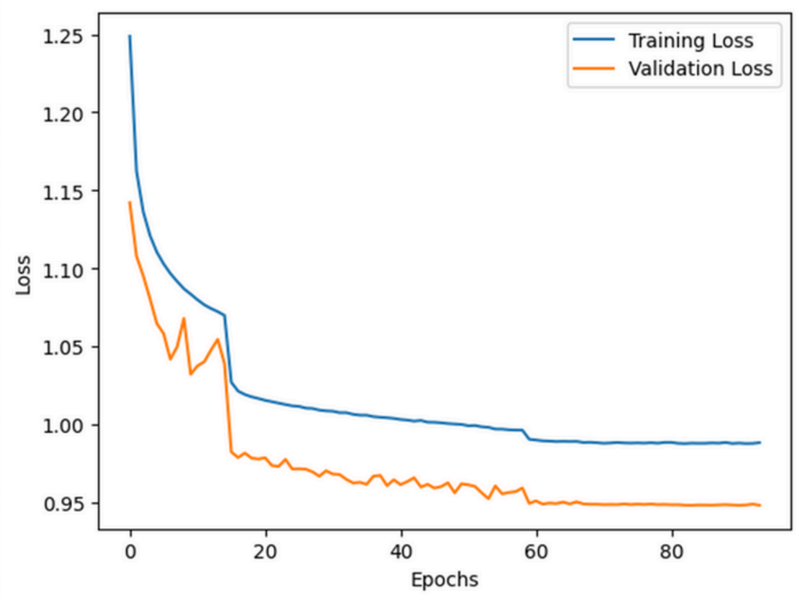
MSE



MAE



Huber



CALLBACKS

- Early Stopping (10 epochs)
- Model checkpoint (min val_loss)
- Reduce LR (5 epochs, factor=0.1)

```
[ ] # 1. Early Stopping: Stops training when validation loss doesn't improve.
    early_stopping = EarlyStopping(
        monitor='val_loss',       # Metric to monitor
        patience=10,              # Number of epochs to wait for improvement
        restore_best_weights=True # Rollback to the best model weights
    )

    # 2. Model Checkpoint: Saves the best model during training.
    model_checkpoint = ModelCheckpoint(
        'nn_best_model.keras',    # Filepath to save the model
        monitor='val_loss',       # Metric to monitor
        save_best_only=True,      # Save only the best model
        mode='min',               # Minimize validation loss
        verbose=1                 # Show a message when saving the model
    )

    # 3. Learning Rate Scheduler: Reduce learning rate when validation loss plateaus.
    reduce_lr = ReduceLROnPlateau(
        monitor='val_loss',       # Metric to monitor
        factor=0.1,               # Factor by which to reduce the learning rate
        patience=5,               # Number of epochs to wait before reducing
        min_lr=1e-6,              # Lower bound for learning rate
        verbose=1                 # Show a message when reducing the learning rate
    )
```

RESULTS

Rider 1

No significant improvement.

LR model: 29.59
NN model: 30.03

Rider 3

Significant improvement.

LR model: 15.35
NN model: 11.85

Rider 7

Significant improvement.

LR model: 26.02
NN model: 13.48

	file	last_time	last_time_pred	percentage_diff
8	lr_f114.csv	5553	5600.173762	0.849518
7	lr_f144.csv	6864	6336.960079	-7.678321
4	lr_f234.csv	4294	3540.409084	-17.549858
5	lr_f24.csv	8877	9524.708456	7.296479
0	lr_f437.csv	1138	766.798228	-32.618785
1	lr_f47.csv	5492	5804.706142	5.693848
6	lr_f537.csv	4311	4608.122235	6.892188
9	lr_f559.csv	1412	980.967598	-30.526374
2	lr_f566.csv	2290	1332.727136	-41.802308
3	lr_f647.csv	5229	5369.212654	2.681443

Absolute percentage difference in LR model: 15.358912331780052

*Example of results for Rider 3

	file	last_time	last_time_pred	percentage_diff
6	nn_f114.csv	5553	5542.771289	-0.184202
0	nn_f144.csv	6864	6394.234106	-6.843909
8	nn_f234.csv	4294	3698.357485	-13.871507
1	nn_f24.csv	8877	8618.184727	-2.915571
4	nn_f437.csv	1138	830.282152	-27.040233
9	nn_f47.csv	5492	5683.119669	3.479965
3	nn_f537.csv	4311	4450.769626	3.242163
5	nn_f559.csv	1412	1140.943112	-19.196663
7	nn_f566.csv	2290	1632.642627	-28.705562
2	nn_f647.csv	5229	4544.694781	-13.086732

Absolute percentage difference in NN model: 11.856650636508206

PREDICTION

Models can be used to predict rider's performance. Results are:

- Individualised
- Terrain dependent
- Reasonable (they obey laws of physics)

	file	real_r3_time	pred_r1_time	pred_r3_time	pred_r7_time
4	nn_f114.csv	5553	4293.671874	5542.771289	5486.792463
1	nn_f144.csv	6864	5108.243633	6394.234110	6381.064014
8	nn_f234.csv	4294	3239.025060	3698.357484	3387.897096
3	nn_f24.csv	8877	6077.393348	8618.184748	9520.767362
6	nn_f437.csv	1138	566.295689	830.282149	549.300692
5	nn_f47.csv	5492	3636.368637	5683.119665	5761.521415
9	nn_f537.csv	4311	3568.671488	4450.769637	4541.254464
0	nn_f559.csv	1412	1141.603622	1140.943115	805.177965
2	nn_f566.csv	2290	1057.814223	1632.642632	1048.191567
7	nn_f647.csv	5229	3696.159426	4544.694773	4048.922932

CHALLENGES

Large Dataset:

- High computational resources and significant time required for EDA and preprocessing due to the dataset's size.
- Each rider contributed up to 700 GPX files, with each file containing over 5,000 rows.

Data Cleaning Issues:

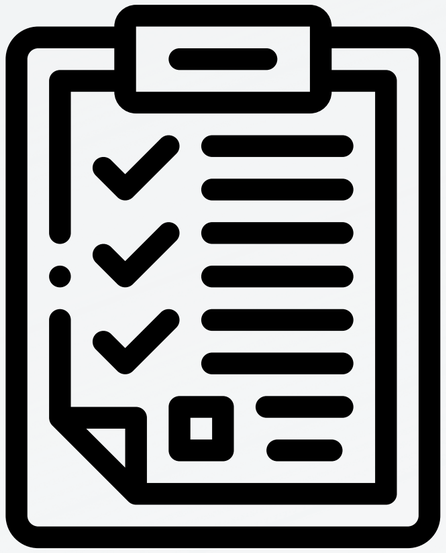
- Identified and filtered out errors data points (e.g., speeds of 700m/s).
- Inconsistent data quality across riders added complexity.

Feature Selection & Tuning:

- Initial model training included multiple features, but key features like slope, angle, and distance proved most effective.
- Extracting relevant features from raw data required meticulous analysis.

Discussion

- Initially, the model struggled to perform well, so we revised our approach to predict speed instead of directly predicting time.



- Using a neural network with multiple layers and dropout techniques, we built a model capable of learning complex patterns while avoiding overfitting.

- Techniques like early stopping and learning rate adjustments helped refine the model and improve performance over time.

- Testing the model with real-world cyclist data confirmed its ability to predict speed with low error rates.

- The model’s success demonstrates its potential to help riders plan their trips more effectively by estimating time based on speed

CONCLUSION AND FUTURE WORK

Successfully developed a model to predict cyclists' speed with high accuracy.

Solved a valuable problem for cyclists and mountaineers, enabling better planning and estimation of travel time.

Integrate weather data via APIs, as weather significantly impacts travel speed and time.

Collect additional data points for improved accuracy and robustness.

Explore real-time prediction capabilities for dynamic trip planning.


Q&A