Collision Prediction Model Definition and Raw Data Requirements

Russell Burdt

4/27/2021

The collision prediction model is designed to predict any collision in a subsequent calendar month for an individual vehicle. The model is based on consistent metrics from several raw data sources. A previous tech memo [*Baseline Collision Model for Munich-Re Collaboration*] described an earlier collision prediction model with a slightly different model definition. This tech memo clearly describes the updated model definition and provides precise details regarding raw data requirements for a model based on appx 1M vehicle evaluations. The intent of the precise description of raw data requirements is to identify a consistent process to extract the requisite data and to motivate enhancements to data architecture if needed.

Collision Prediction Model Definition

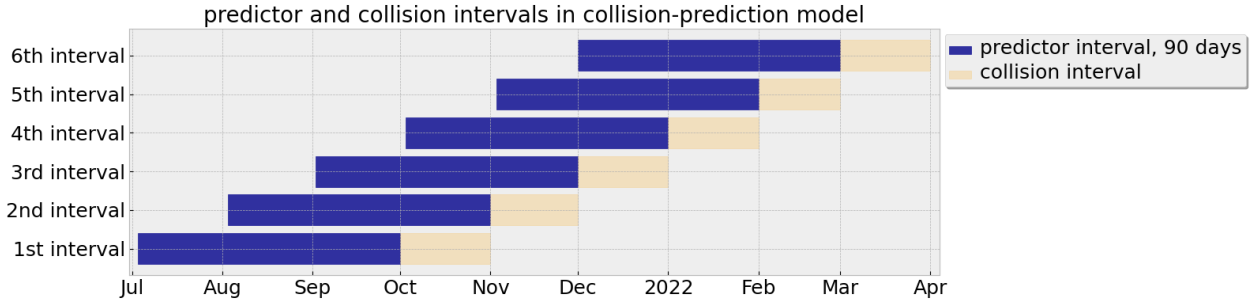
A starting point for the collision prediction model is the metadata below. The *desc* and *devices* fields define a population of vehicles – all vehicles in Distribution, Transit, and Freight/Trucking industries with SF300 or SF64 devices (appx 300k vehicles).

desc	all of Distribution, Transit, Freight/Trucking
collision intervals 2021	Oct,Nov,Dec
collision intervals 2022	Jan,Feb,Mar
predictor interval days	90
devices	ER-SF300,ER-SF64

The key idea of the model is to use data in a *predictor-interval* (*time0* to *time1*) to make predictions of any collision in a *collision-interval* (*time1* to *time2*). Note the shared boundary. The predictor-interval is always 90 days and the collision-interval is always a full calendar month.

time0	time1	time2
2021-07-03	2021-10-01	2021-11-01
2021-08-03	2021-11-01	2021-12-01
2021-09-02	2021-12-01	2022-01-01
2021-10-03	2022-01-01	2022-02-01
2021-11-03	2022-02-01	2022-03-01
2021-12-01	2022-03-01	2022-04-01

The table and visualization represent predictor and collision intervals defined by the collision prediction model metadata above. Data over multiple intervals are used for training and validating the collision prediction model. In production, *time1* of the model could be the present time and the model would then offer predictions of any collision for a population of vehicles (may be different than the training population) over the subsequent calendar month.



An individual positive row in the collision prediction model population is to the right. A row is a *VehicleId* associated with an *EventRecorderId* over the full duration of a single predictor and subsequent collision interval. Vehicles where *EventRecorderId* changes over the duration of a predictor/collision interval are not included in the model.

```
VehicleId           1C00FFFF-59AE-49C9-9DC0-4663F0800000
EventRecorderId     1C00FFFF-2688-34E6-9E9F-4663F0800000
Model                                             ER-SF64
GroupId             1C00FFFF-F993-92B2-E71C-5D43E7800000
GroupName                        Morristown Express 6336
CompanyName                    LCL Bulk Transport, Inc.
IndustryDesc                                Distribution
CompanyId                                           3809
time0                                2021-07-03 00:00:00
time1                                2021-10-01 00:00:00
time2                                2021-11-01 00:00:00
collision-47                                        True
collision-47-idx                                [17801]
```

The row is positive as there is a recorded collision (behavior-id of 47) in the collision interval. Note there may be more than one collision in the collision interval for a positive row. Data for the collision are to the right, where the associated row is referenced by the field *collision-47-idx* in the population table.

```
In [40]: dp.loc[17801]
Out[40]:
VehicleId               1C00FFFF-59AE-49C9-9DC0-4663F0800000
RecordDate                                2021-10-31 08:10:56
Latitude                                           40.493467
Longitude                                         -86.86184
EventId                 1C00FFFF-CA8B-883F-3AD8-4663F0800000
CustomerEventIdString                                XJDM04492
BehaviorId                                              47
timezone                          America/Indiana/Indianapolis
localtime                            2021-10-31 04:10:56-04:00
state                                               Indiana
country                                                  US
```

Negative rows in the collision model population are those with no recorded collisions in the collision interval; most rows are negative. For the collision model population based on appx 1M vehicle evaluations appx 0.7% of rows reference any collision in the collision interval.

Collision Prediction Model Raw Data Architecture

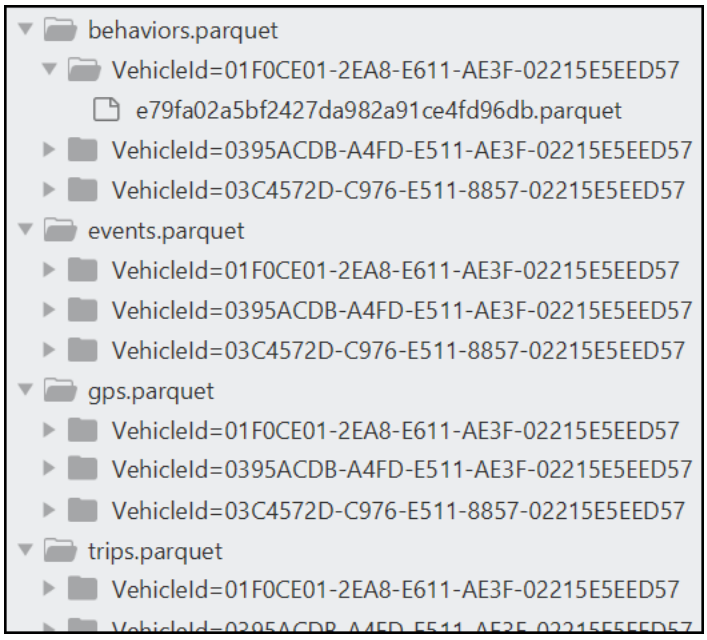
Collision predictions are based on aggregated metrics for individual vehicles over the predictor interval, and there may be several hundred metrics, eg

- number of hard braking events
- number of hard braking events on all Mondays
- duration of contiguous GPS records that indicate speed greater than 0.1mph
- average speed of all accelerometer events between 1900 and 2300 hours local time
- standard deviation of all trip distances
- number of events where food/drink distractions are observed
- average collision-detection model score of all accelerometer events

These aggregated metrics are derived from the following database-schema-tables:

Database	Schema	Table(s)	Data Source(s)
EDW	flat	Events	events / behaviors
EDW	gps	Trips	trips
EDW	ml	Dce / ModelRequest / ModelResponse	dce-scores
EDW	hs	EventRecorderFiles / EventRecorderFileTriggers	triggers
Snowflake	GPS	GPS_ENRICHED	gps

The collision prediction model extracts raw data from the above tables to an intermediate storage layer that enables very fast subsequent processing. The raw data storage layer needs to support very fast processing as metric definitions are rapidly iterated. Right now the intermediate storage layer is either a Parquet dataset on Elastic Block Storage or a Postgres database, all on AWS. It would not be practical to extract aggregated metrics directly from the raw data sources as that would not support rapid iteration of metrics.



The screenshot to the right represents raw data stored as a Parquet dataset by data source name, eg *events.parquet*. All data source names are referenced in the above table. Data are partitioned by *VehicleId* or *EventRecorderId*. Only three partitions are in the screenshot however a real Parquet dataset may contain 100s of thousands of partitions.

Collision Prediction Model Raw Data for 250k Vehicle Evaluations

A dataset representing appx 250k vehicle evaluations has already been created, and the initial collision prediction model is trained and validated on these data. Data volume and number of partitions in the Parquet dataset by data source name is below.

```
behaviors.parquet - GB, num of partitions      1.7GB, 109369
dce_scores.parquet - GB, num of partitions     0.9GB, 98260
events.parquet - GB, num of partitions         2.8GB, 115039
gps.parquet - GB, num of partitions          179.3GB, 115979
triggers.parquet - GB, num of partitions      32.4GB, 116302
trips.parquet - GB, num of partitions          2.4GB, 114093
```

There were two main issues encountered in extracting these raw data to Parquet datasets:

- Extracting the 180GB of GPS data from Snowflake incurred a non-negligible cost
- **Extracting the 32GB from EDW hs.EventRecorder* tables was painfully slow!!!** Data were extracted over weeks and in several instances other processes on EDW were interrupted by the query load. Data-teams are well aware of these issues.

A next step in the development of the collision prediction model is to build a model with more data (appx 1M vehicle evaluations). Data volume will grow proportionally, ie by a factor of 4 with respect to the above table. Lytx needs a method to efficiently extract these raw data, in less than a week to enable rapid development of the model and at a reasonable cost. The method may need to run many more times as further model development proceeds.

Appendix – Raw data queries

This appendix provides the exact raw data queries used by the collision prediction model. In all cases, data are extracted within a time window for a list of vehicle-ids or event-recorder-ids. The time-window is all of the predictor intervals for the set of vehicles / event-recorders. The queries are created via Python, where string-formatting is used to provide the time-window and list of vehicle-ids or event-recorder-ids. Currently, the length of the list of vehicle-ids or event-recorder-ids is 200 to 1000, so in order to acquire the full raw data (appx 100 thousand partitions), the same query is broken up accordingly. The current infrastructure also supports concurrent queries to EDW/Snowflake, which in some cases can interrupt other processes.

Query for events data source

```
SELECT
    E.VehicleId,
    E.RecordDate,
    E.Latitude,
    E.Longitude,
    E.EventId,
    E.EventRecorderId,
    E.EventRecorderFileId,
    E.SpeedAtTrigger,
    E.EventTriggerTypeId AS NameId,
    E.EventTriggerSubTypeId AS SubId,
    T.Name,
    E.EventFileName,
    CASE WHEN E.EventFileName LIKE 'event%'
        THEN 'lytx-amlnas-us-west-2'
        ELSE NULL
    END AS s3_bucket_name,
    CASE WHEN E.EventFileName LIKE 'event%'
        THEN REPLACE(REPLACE(CONCAT(E.EventFilePath, '/',
CAST(CONCAT('<t>', REPLACE(E.EventFileName, '_', '</t><t>'), '</t>') AS
XML).value('/t[2]', 'varchar(100)'), '/', E.EventFileName),
'\\\\drivecam.net', 'dce-files'), '\\', '/')
        ELSE NULL
    END AS s3_key,
    CASE WHEN E.EventFileName LIKE 'event%'
        THEN
        REPLACE(REPLACE(CONCAT(E.EventFilePath, '/', CAST(CONCAT('<t>',
REPLACE(E.EventFileName, '_', '</t><t>'), '</t>') AS XML).value('/t[2]',
'varchar(100)'), '/', E.EventFileName), '\\\\drivecam.net', 's3://lytx-
amlnas-us-west-2/dce-files'), '\\', '/')
    ELSE NULL
    END AS s3_uri
FROM flat.Events AS E
    LEFT JOIN hs.EventTriggerTypes_i18n AS T
    ON T.Id = E.EventTriggerTypeId
WHERE E.Deleted=0
AND E.RecordDate BETWEEN '{df['time0'].min()}' AND '{df['time1'].max()}'
AND E.VehicleId IN ({vstr})
```

Query for behaviors data source

```
WITH behaviors AS (
    SELECT
        VehicleId,
        RecordDate,
        Latitude,
        Longitude,
        EventId,
        EventRecorderId,
        value AS BehaviorId,
        CASE WHEN EventFileName LIKE 'event%'
            THEN 'lytx-amlnas-us-west-2'
            ELSE NULL
        END AS s3_bucket_name,
        CASE WHEN EventFileName LIKE 'event%'
            THEN REPLACE(REPLACE(CONCAT(EventFilePath, '/',
CAST(CONCAT('<t>', REPLACE(EventFileName, '_', '</t><t>'), '</t>') AS
XML).value('/t[2]', 'varchar(100)'), '/', EventFileName),
'\\\\drivecam.net', 'dce-files'), '\\', '/')
            ELSE NULL
        END AS s3_key,
        CASE WHEN EventFileName LIKE 'event%'
            THEN
            REPLACE(REPLACE(CONCAT(EventFilePath, '/', CAST(CONCAT('<t>',
REPLACE(EventFileName, '_', '</t><t>'), '</t>') AS XML).value('/t[2]',
'varchar(100)'), '/', EventFileName), '\\\\drivecam.net', 's3://lytx-
amlnas-us-west-2/dce-files'), '\\', '/')
        ELSE NULL
        END AS s3_uri
    FROM flat.Events
        CROSS APPLY STRING_SPLIT(COALESCE(BehaviourStringIds, '-1'), ',')
    WHERE value <> -1
    AND Deleted = 0
    AND RecordDate BETWEEN '{df['time0'].min()}' AND '{df['time1'].max()}'
    AND VehicleId IN ({vstr}))
SELECT
    behaviors.VehicleId,
    behaviors.RecordDate,
    behaviors.Latitude,
    behaviors.Longitude,
    behaviors.EventId,
    behaviors.EventRecorderId,
    behaviors.BehaviorId AS NameId,
    hsb.Name,
    behaviors.s3_bucket_name,
    behaviors.s3_key,
    behaviors.s3_uri
FROM behaviors
    LEFT JOIN hs.Behaviors_i18n AS hsb
    ON behaviors.BehaviorId = hsb.Id
```

Query for trips data source

```
SELECT
    VehicleId,
    StartPositionTimeUTC AS time0,
    EndPositionTimeUTC AS time1,
    Distance,
    TripPointCount
FROM gps.Trips
WHERE StartPositionTimeUTC > '{df['time0'].min()}'
AND EndPositionTimeUTC < '{df['time1'].max()}'
AND VehicleId IN ({vstr})
```

Query for dce-scores data source

```
SELECT
    E.VehicleId,
    E.RecordDate,
    E.EventId,
    E.Latitude,
    E.Longitude,
    RE.ModelValue
FROM flat.Events AS E
    INNER JOIN ml.Dce AS DCE
    ON DCE.EventId = E.EventId
    INNER JOIN ml.ModelRequest AS RQ
    ON RQ.DceId = DCE.DceId
    INNER JOIN ml.ModelResponse AS RE
    ON RE.ModelRequestId = RQ.ModelRequestId
    AND RE.ModelKey = 'collision'
    AND RQ.ModelId = 4
WHERE E.Deleted=0
AND E.EventTriggerTypeId in (27,30,31)
AND E.RecordDate BETWEEN '{df['time0'].min()}' AND '{df['time1'].max()}'
AND E.VehicleId IN ({vstr})
```

Query for GPS data source

```
SELECT
    UPPER(VEHICLE_ID) AS VEHICLEID,
    TS_SEC,
    LATITUDE,
    LONGITUDE
FROM GPS.GPS_ENRICHED
WHERE TS_SEC BETWEEN {time0} AND {time1}
AND (VEHICLE_ID IN ({vstr_lower}) OR VEHICLE_ID IN ({vstr_upper}))
```

Query for triggers data source

```
SELECT
    ERF.EventRecorderId,
    ERF.EventRecorderFileId,
    ERF.CreationDate,
    ERF.FileName,
    ERF.EventTriggerTypeId,
    ERFT.TriggerTime,
    ERFT.Position.Lat AS Latitude,
    ERFT.Position.Long as Longitude,
    ERFT.ForwardExtremeAcceleration,
    ERFT.SpeedAtTrigger,
    ERFT.PostedSpeedLimit
FROM hs.EventRecorderFiles AS ERF
    LEFT JOIN hs.EventRecorderFileTriggers AS ERFT
    ON ERFT.EventRecorderFileId = ERF.EventRecorderFileId
WHERE ERF.EventRecorderId IN ({ers})
AND ERFT.TriggerTime BETWEEN '{tmin}' AND '{tmax}'
```