Baseline Collision Model for Munich-Re Collaboration

Russell Burdt

11/29/2021

A collaboration with the insurance provider Munich-Re was agreed in Nov 2021 to score the collision risk of vehicles in a dataset provided by Munich-Re. This document precedes delivery of the dataset by Munich-Re and explores a baseline collision model based on data we expect to be within the Munich-Re dataset, and intentionally excludes data we do not expect. The expected data sources include *EDW gps.Trips* and *flat.Events*, and notably exclude raw GPS data such as the *Snowflake GPS_ENRICHED* data source. The baseline model is a first attempt at a collision model, to establish the model definition, evaluation metrics, and initial performance. Subsequent models are likely to improve with respect to the baseline performance.

Model Definition

The row definition in the collision model is a fixed time interval for an individual vehicle, eg VehicleId 9100FFFF-48A9-CB63-3C01-A8A3E3070000 between 8/1/2021 and 9/1/2021. The end of the fixed time interval will represent either

- the moment of a collision for the same vehicle (positive instance), OR
- a moment in time when there is no collision within some subsequent number of days (eg 20 days) for the same vehicle (negative instance).

The model is then a binary classification model to predict the collision indication for a specific population of vehicles (eg all vehicles in the Distribution industry). The duration of the time interval for all rows will be identical (eg 30 days), however the specific interval boundaries (start and end time) will vary because a vehicle may experience a collision at any moment in time. The full range of the time interval end time for all rows will be some number of days preceding a fixed date, eg 365 days before 11/1/2021. Finally, the collision definition may be one of

- All events with a behavior ID of 47 (collision)
- All events with a behavior ID of 45 (near collision, unavoidable), 46 (near collision), or 47

Python and SQL Implementation of Model Definition

The model definition can be parameterized by a population configuration and a time-window configuration. The population configuration defines the metadata filters for the set of vehicles to include in the model, and other misc filters. The baseline model excludes test devices and unregistered devices, and includes only SF300 devices as an attempt to control for device configuration. A query to return vehicle metadata based on these filters, and for a population that includes all vehicles in the *Distribution, Concrete,* and *Freight/Trucking* industries is below. The query executed on 11/23/2021 returned 176,674 vehicles.

```
SELECT
    D.VehicleId,
    D.SerialNumber,
    C.CompanyName,
    C.IndustryDesc,
    C.IndustrySectorDesc
FROM flat.Companies AS C
    LEFT JOIN flat.Devices AS D
    ON C.CompanyId = D.CompanyId
WHERE C.CompanyName <> 'DriveCam DC4DC Test Co'
AND D.Model = 'ER-SF300'
AND D.VehicleId <> '00000000-0000-0000-0000-000000000000'
AND IndustryDesc IN ('Distribution','Concrete','Freight/Trucking')
```

The time-window configuration as implemented in Python is below, where the individual items parameterize the model definition. As described previously, each row is based on a time-interval for an individual vehicle. The end of the time-interval for all rows may be between `T1` and `WINDOW_POSITIVE_CLASS_DAYS` prior to `T1`, eg between 11/1/2020 to 11/1/2021. The duration of each time-interval is `WINDOW_DAYS`, eg 30 days. The positive class definition is parameterized in `POSITIVE_CLASS_BEHAVIOR_IDS`, eg only events with a behavior ID of 47.

```
T1 = '2021-11-01'
WINDOW_DAYS = 30
WINDOW_POSITIVE_CLASS_DAYS = 365
WINDOW_NEGATIVE_CLASS_DAYS = 20
POSITIVE_CLASS_BEHAVIOR_IDS = [47]
```

A first step in defining specific time-intervals for individual vehicles is to identify all collisions as defined in the previous configuration. The query below identifies all collisions for all registered vehicles in the requisite time window.

```
SELECT
    VehicleId,
    RecordDate,
    value AS BehaviorId
FROM flat.Events
    CROSS APPLY STRING_SPLIT(COALESCE(BehaviourStringIds, '-1'), ',')
WHERE RecordDate BETWEEN '11-01-2020 00:00:00' AND '11-01-2021 00:00:00'
AND VehicleId <> '00000000-0000-0000-0000-000000000000'
AND Deleted=0
AND value IN (47)
```

A Python post-processing step filters results of the above query to vehicles in the population defined previously; the result is 11,346 collision instances. A single vehicle may contribute more than one collision in the time window of the previous query. In this case there are 10,269 vehicles that contribute the 11,346 collision instances. Each collision instance is associated with a timestamp, and the corresponding time-interval is defined relative to that timestamp. For example, the table below represents 5x positive instances. Note the same vehicle-id appears in two distinct rows, however each of those rows is associated with a unique 30 day time-interval.

VehicleId	time0	time1	outcome
9100FFFF-48A9-D463-C2FA-3A63F3FF0000	2021-06-03 18:59:29	2021-07-03 18:59:29	47
9100FFFF-48A9-D363-DD91-2143E4FF0000	2021-01-08 11:51:02	2021-02-07 11:51:02	47
9100FFFF-48A9-D363-DD91-2143E4FF0000	2020-10-03 12:27:01	2020-11-02 12:27:01	47
9100FFFF-48A9-D463-C082-3A63F3FF0000	2021-04-29 07:26:47	2021-05-29 07:26:47	47
9100FFFF-48A9-D463-C2F8-3A63F3FF0000	2020-11-24 19:18:28	2020-12-24 19:18:28	47

A subsequent post-processing step identifies vehicles in the original population that were not involved in a collision in the 365 day time window; in this case there are 166,405 such vehicles. One random time-interval is selected for each of those vehicles, where the possible end time of the interval is defined by the time-window configuration, in this case between 11/1/2020 and 10/12/2021 (ie 20 days before 11/1/2021 as set by `WINDOW_NEGATIVE_CLASS_DAYS`). This implementation means there is no collision for these instances during the time interval and for at least 20 days after the time interval. The table below represents 5x negative instances.

VehicleId	time0	time1	outcome
1C00FFFF-59AE-49C9-C482-4663F0800000	2021-06-12	2021-07-12	0
9100FFFF-48A9-D863-4C20-8E63F0480000	2021-01-31	2021-03-02	0
9100FFFF-48A9-D863-4BCD-8E63F0480000	2021-07-30	2021-08-29	0
1C00FFFF-59AE-49C9-C3F3-4663F0800000	2021-05-28	2021-06-27	0
1C00FFFF-59AE-49C9-C47D-4663F0800000	2020-12-09	2021-01-08	0

At this point the population statistics can be summarized concisely, as seen to the right. The table summarizes metrics described previously, as well as precise counts for the number of vehicles with recurring collisions. Note that the number of vehicle/time-windows (ie an individual vehicle-id and a 30 day time-window) may exceed the number of unique vehicles. A vehicle/time-window is subsequently referred to as a *vehicle history*.

num vehicle/time-windows	177751
num vehicles	176674
num positive instances	11346
percentage positive instances, %	6.38
num negative instances	166405
percentage negative instances, %	93.62
num vehicle-id with 1 positive instance	9298
num vehicle-id with 2 positive instance	881
num vehicle-id with 3 positive instance	76
num vehicle-id with 4 positive instance	12
num vehicle-id with 5 positive instance	2
num vehicle-id with 1 negative instance	166405

One deficiency with this model implementation is that by definition all vehicle histories with any previous collision are positive instances. So any model feature that represents the 'number of previous collisions' will always be associated with a positive instance when the feature value is greater than zero. The issue will be resolved in a subsequent model implementation.

Raw Data for Each Vehicle History

The previous section describes how a population configuration and time-window configuration are used to create a table of vehicle histories, where each vehicle history either ends in a collision, or does not end in a collision (with no collision for 20 subsequent days). This section describes raw data collection for all vehicle histories.

Raw data collection for the baseline model uses all records in *EDW gps.Trips* and *flat.Events* over each vehicle history. Information in *flat.Events* is split into data for events and separate data for behaviors. The query below extracts raw events data for a single vehicle history.

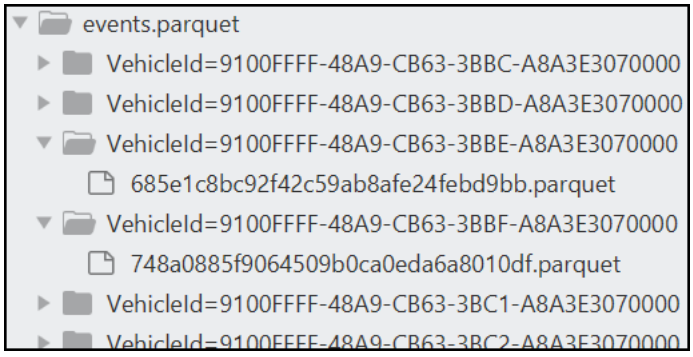
```
SELECT
    E.VehicleId,
    E.RecordDate,
    E.Latitude,
    E.Longitude,
    E.EventId,
    E.EventTriggerTypeId AS NameId,
    E.EventTriggerSubTypeId AS SubId,
    T.Name
FROM flat.Events AS E
    LEFT JOIN hs.EventTriggerTypes_i18n AS T
    ON T.Id = E.EventTriggerTypeId
WHERE E.Deleted=0
AND E.RecordDate BETWEEN '2020-10-02 00:00:00' AND '2020-11-01 00:00:00'
AND E.VehicleId IN ('1C00FFFF-59AE-4AC9-063D-4663F0800000')
```

A separate query for each vehicle history (as in the above query) is not implemented because of slow execution time for 100s of thousands of unique vehicle histories as in the baseline model. High-level details of the algorithm developed to query raw data for all vehicle histories (where each vehicle history may have a unique time interval) is as follows:

- Identify sets of vehicle histories with similar start and end time, eg 200 vehicles in a set
- Query for raw data over the full range of all time intervals in each set of vehicles
- Remove data outside of individual vehicle histories in a post-processing step
- Execute all queries for all sets of vehicles concurrently (eg using the Python dask library)

The algorithm extracts the same data as in the query above, except it does so efficiently for all unique vehicle histories. Further details of the algorithm are not included in this document. The same algorithm is applied to extract raw data from other data sources over the 100s of thousands of unique vehicle histories in the baseline model.

Data storage is implemented using the Parquet format. Each raw data source becomes an individual Parquet dataset partitioned by VehicleId. A screenshot of the *events.parquet* dataset is to the right. With data in this format, Spark can be used subsequently for parallel data-processing. There are appx 2.5M total rows in the events Parquet dataset for the baseline model.



The same algorithm and Parquet data storage format are used to create *behaviors.parquet* and *trips.parquet* datasets, representing all raw data used in the baseline model (together with *events.parquet*). The two queries below represent raw data collection over a single vehicle history for behaviors and trips, respectively. The total number of rows for the behaviors and trips datasets are appx 1.3M and 11.8M, respectively. The total data size for all raw data in the baseline model is appx 5.4GB.

```
WITH behaviors AS (
    SELECT
        VehicleId,
        RecordDate,
        Latitude,
        Longitude,
        EventId,
        value AS BehaviorId
    FROM flat.Events
        CROSS APPLY STRING_SPLIT(COALESCE(BehaviourStringIds, '-1'), ',')
    WHERE value <> -1
    AND Deleted = 0
    AND RecordDate BETWEEN '2020-10-02 00:00:00' AND '2020-11-01 00:00:00'
    AND VehicleId IN ('1C00FFFF-59AE-4AC9-063D-4663F0800000'))
SELECT
    behaviors.VehicleId,
    behaviors.RecordDate,
    behaviors.Latitude,
    behaviors.Longitude,
    behaviors.EventId,
    behaviors.BehaviorId AS NameId,
    hsb.Name
FROM behaviors
    LEFT JOIN hs.Behaviors_i18n AS hsb
    ON behaviors.BehaviorId = hsb.Id
```

```
SELECT
    VehicleId,
    StartPositionTimeUTC AS time0,
    EndPositionTimeUTC AS time1,
    Distance,
    TripPointCount
FROM gps.Trips
WHERE StartPositionTimeUTC > '2020-10-02 00:00:00'
AND EndPositionTimeUTC < '2020-11-01 00:00:00'
AND VehicleId IN ('1C00FFFF-59AE-4AC9-063D-4663F0800000')
```

Metrics for Each Vehicle History – Implementation and Analysis

The previous section describes how raw data representing all event, behavior, and trip records over all vehicle histories are collected and organized in a 5.4GB Parquet dataset. This section describes the consistent metrics that are extracted from raw data for each vehicle history.

The PySpark SQL library is used to read raw data in Parquet datasets and efficiently extract consistent metrics using all available CPUs. For example, the code example below represents minimal steps to read the *trips.parquet* dataset and count all rows, identifying 11.8M rows as previously described. The only required PySpark configuration used for the baseline model is to set driver memory according to the processing task.

```python
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()
trips = r'/mnt/home/russell.burdt/data/collision-study/dcft/trips.parquet'
trips = spark.read.parquet(trips)
trips.createTempView('trips')
df = spark.sql(f'SELECT COUNT(*) FROM trips').toPandas()
```

```
In [9]: df
Out[9]:
   count(1)
0  11780045
```

The baseline model intentionally uses very basic metrics derived from raw data. These include standard trip metrics, count of individual/all events, and count of individual/all behaviors. More advanced metrics based on the same raw data and/or other sources are reserved for subsequent models.

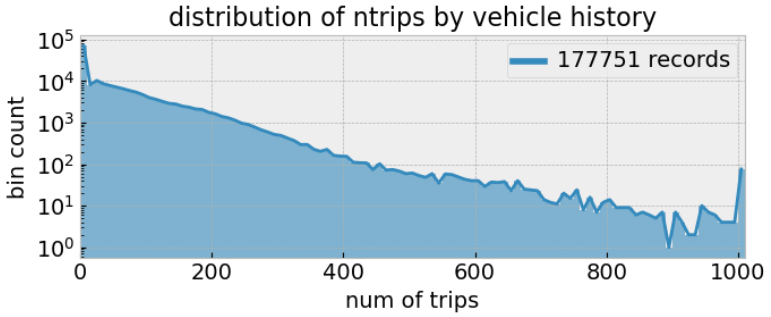
Metrics extraction uses a subset of the vehicle histories table, where a row-id column has been added; the screenshot to the right is 5x random rows.

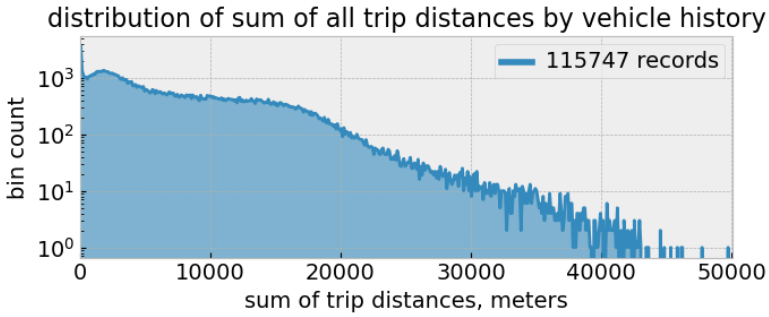
rid	VehicleId	time0	time1
0	9100FFFF-48A9-D463-AEE1-3A63F3FF0000	1602666659	1605258659
1	9100FFFF-48A9-D463-AF42-3A63F3FF0000	1621975839	1624567839
2	9100FFFF-48A9-D463-AF56-3A63F3FF0000	1616581894	1619173894
3	9100FFFF-48A9-D463-AF56-3A63F3FF0000	1616582505	1619174505
4	9100FFFF-48A9-D463-AF58-3A63F3FF0000	1626834003	1629426003
5	9100FFFF-48A9-D463-AF5F-3A63F3FF0000	1620290462	1622882462

The PySpark SQL code to the right is the exact query implemented to extract standard trip metrics by row-id. Note that valid trips are those with entire trip bounds (start and end time) contained within the time-interval of the vehicle history.

```python
dt = spark.sql(f"""
    SELECT
        df.rid,
        COUNT(*) AS ntrips,
        SUM(trips.Distance) AS trips_distance_sum,
        AVG(trips.Distance) AS trips_distance_mean,
        STDDEV(trips.Distance) AS trips_distance_std
    FROM df JOIN trips
        ON trips.VehicleId = df.VehicleId
        AND trips.TS_SEC0 > df.time0
        AND trips.TS_SEC1 < df.time1
    GROUP BY df.rid""").toPandas()
```

Distributions of two of the standard trip metrics for all vehicle histories are below. Note that appx 35% of all vehicle histories in the baseline model do not contain any trips. The vehicle histories that are negative instances (no collision) may not contain any trips if the vehicle was unused in the time-window or if use was not registered in any trips. The vehicle histories that are positive instances (ie with collisions) would have been used prior to the collision, and appx 95% of the positive instances contain one or more trips. The case of vehicle histories that are positive instances with no registered trips was investigated – the vehicle must have an ignition signal to the Event Recorder to register a trip which may not always be present in all vehicles.

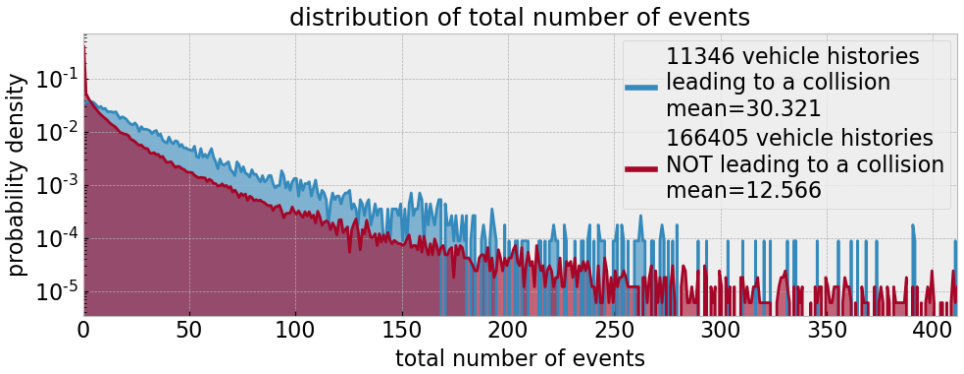




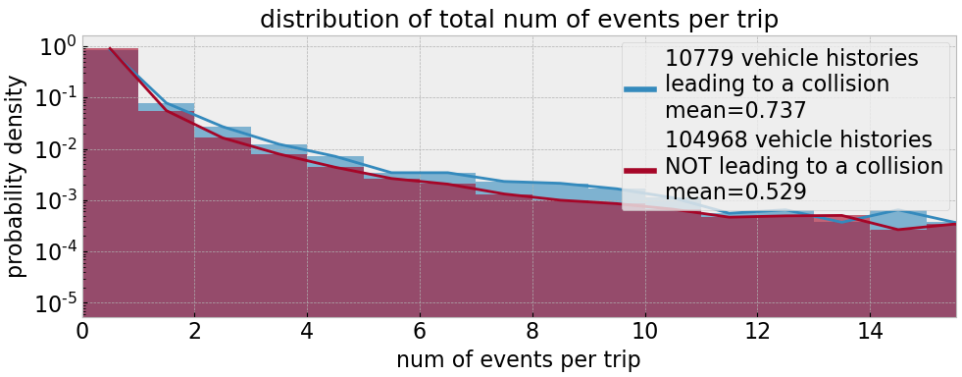
In addition to the four standard trip metrics (num of trips, sum/mean/stdev of trip distances), the total count of events, and total count of behaviors are used as metrics for the baseline model. The PySpark SQL code to the right extracts the count of events for each vehicle history. Similar code extracts the count of behaviors for each vehicle history.

```
de0 = spark.sql(f"""
    SELECT
        df.rid,
        COUNT(*) AS nevents
    FROM df JOIN events
        ON events.VehicleId = df.VehicleId
        AND events.TS_SEC >= df.time0
        AND events.TS_SEC < df.time1
    GROUP BY df.rid""").toPandas()
```

Each vehicle history may have a unique count of events, and it is useful to visualize distributions of count of events for positive and negative vehicle histories. The figure below represents both distributions, where the y-axis is probability density to account for the difference in population size. For example, the probability density representing one positive vehicle history out of 11,346 is appx 10^{-4}. The same for one negative vehicle history is appx 10^{-5}. Bin width in both distributions in 1 event. Note the significant difference in the first bin – appx 40% of negative vehicle histories do not have any events, whereas appx 3% of positive vehicle histories do not have any events. Also note the significant difference in the mean of each distribution – appx 30 events vs 13 events (on average) for positive vs negative vehicle histories, respectively. For this reason the number of events by vehicle history may be a useful feature for machine learning.



The distribution above indicates an increase in total number of events (on average) for positive vs negative vehicle histories (the difference is statistically significant). As many events are the result of normal everyday vehicle usage (eg accelerometer events), the result may also be interpreted as meaning 'more usage is associated with more collisions'. So it is also useful to visualize distributions of a normalized version of the above metric, eg 'count of events per trip' for positive vs negative vehicle histories; the distributions are below. Note that population size in both distributions have decreased as some positive and negative vehicle histories do not contain any trips. The difference in distribution means is 0.737 events per trip vs 0.529 events per trip (on average) for positive vs negative vehicle histories, respectively. The difference in means was confirmed as statistically significant via a hypothesis test.



In addition to the above metrics (standard trip metrics + count of all events + count of all behaviors) the baseline collision model also uses the count of individual events and same for behaviors. The core code used to extract counts of individual events is to the right, and uses a query and pivot in PySpark SQL as well as some additional post-processing to merge with other metrics. The number of unique events that are counted will depend on the population, as some events may never be triggered in some populations. For the population of the baseline model, there are 21 unique individual events, and 74 unique individual behaviors.

```
de1 = spark.sql(f"""
    SELECT
        CAST(df.rid AS STRING) AS rid,
        CAST(events.NameId AS STRING) AS NameId,
        COUNT(*) AS nevents
    FROM df JOIN events
        ON events.VehicleId = df.VehicleId
        AND events.TS_SEC >= df.time0
        AND events.TS_SEC < df.time1
    GROUP BY df.rid, events.NameId""")
de1 = de1.groupBy('rid').pivot('NameId').max().toPandas()
```

Dataset for Machine Learning

Previous sections desribed how to create a table of vehicle histories, then collect raw data and extract consistent metrics for each vehicle history. The consistent metrics described so far include trip metrics for each vehicle history (num of trips, mean/sum/stdev of all trip distances), count of all events, count of all behaviors, as well as count of each individual event and behavior. The total count of all metrics is 4x trip metrics, 2x count of events/behaviors, 21x count of individual events, and 74x count of behaviors, leading to 101 metrics total.

In addition to the 101 metrics, the baseline collision model also uses 3x binary-encoded industry metrics (one of Concrete, Freight/Trucking, or Distribution), and 4x binary-encoded calendar-quarter metrics (one of Q1/Q2/Q3/Q4) representing the calendar quarter containing the majority of the time interval for each vehicle history. Accordingly, the baseline collision model uses 108 total metrics.

A machine-learning model will require all data to be non-null, which drives a requirement to remove vehicle histories with zero or one trip, as there will be a null ‘standard deviation of trips distance’ for those rows. The final population statistics representing data used in machine learning is to the right. Note the positive instance percentage is appx 10% because many negative instances were removed by the trip filter.

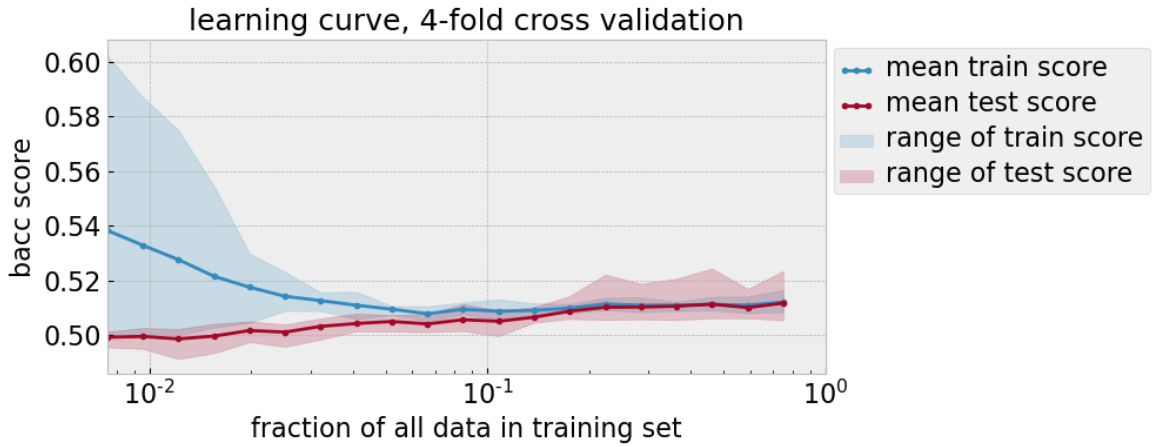
num vehicle/time-windows	114025
num vehicles	113005
num positive instances	10728
percentage positive instances, %	9.41
num negative instances	103297
percentage negative instances, %	90.59
num vehicle-id with 1 positive instance	8790
num vehicle-id with 2 positive instance	830
num vehicle-id with 3 positive instance	76
num vehicle-id with 4 positive instance	10
num vehicle-id with 5 positive instance	2
num vehicle-id with 1 negative instance	103297

Accordingly, the dataset used for the baseline collision model is 114,025 rows and 108 columns, where appx 10% of rows are positive instances.

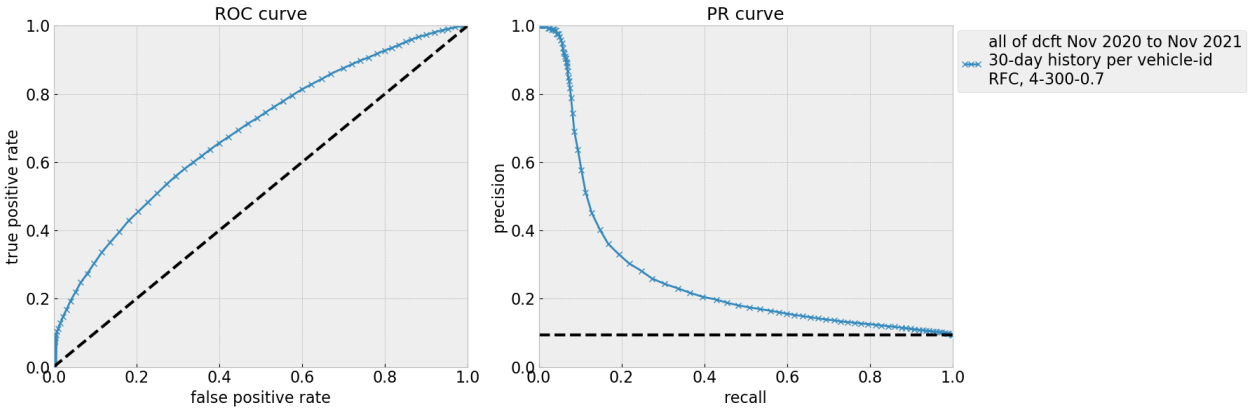
Random Forest Algorithm as Baseline Collision Model

Of the many choices for a machine learning algorithm, the Random Forest algorithm was chosen as this is a non-linear, ensemble algorithm that can identify associations in unscaled data of mixed datatype as in the baseline collision model dataset. The high-level API in scikit-learn was used to run the model.

The Random Forest algorithm requires some hyperparameters to be tuned to avoid over-fitting, ie the max-depth, number of estimators, and positive class weight. Some iteration was used to identify the following settings that avoid overfitting – max-depth of 3, number of estimators of 300, and positive class weight of 70%. Overfitting is validated via the learning curve seen below. The learning curve visualizes a single train / test classification metric as the fraction of training data is varied. Balanced accuracy (mean of true-positive-rate and false-positive-rate) was used to validate same train/test score. Additionally, the slope of the test score is gradually increasing and does not seem to have flattened at max data volume, indicating more data may be useful. Overall, the learning curve indicates the model is not over-fitting (difference in test score vs train score variability is noted but not severe).



The model with same hyperparameters is then fit on all data, and prediction probabilities are extracted. The threshold limit for a positive prediction is then scanned between min and max prediction probability to sweep out an ROC curve and a PR curve. Both are seen below.



The ROC and PR curves of the previous section indicate the range of possible evaluation metrics where the baseline model performs. Note the ideal model sits at the top-left corner of the ROC curve and top-right corner of the PR curve. At very high values of precision (where majority of positive indications are correct) the recall (true-positive-rate, fraction of correct positive indications) is quite low, appx 5%. The objective of subsequent models should be to maintain high precision (above 90%) and gradually increase the recall.

Feature importance is supported by a Random Forest model; and the top 20 most important features are below. The most important feature is number of previous collisions, ie 'nbehaviors-47'. This is not surprising because the model definition embeds that all vehicle histories with any previous collisions are positive instances (a deficiency). The total number of events is the next most important feature, which was previously explored via distributions on its own and shown to be a useful feature. The next few most important features include the count of accelerometer events and various trip metrics, which also makes sense. Interestingly, the 'Freight/Trucking' industry indicator and 'Q4' indicator are top features, which may be related to the unique driving case and general weather patterns in Q4.

```
In [7]: df.columns[np.argsort(model.feature_importances_)[::-1]][:20]
Out[7]:
Index(['nbehaviors-47', 'nevents', 'nevents-30', 'nevents-32', 'nbehaviors',
       'trips_distance_sum', 'trips_distance_std', 'nbehaviors-142',
       'trips_distance_mean', 'Freight/Trucking', 'nevents-27', 'Q4',
       'nbehaviors-59', 'nbehaviors-80', 'nevents-15', 'nbehaviors-69',
       'nbehaviors-87', 'nbehaviors-11', 'nbehaviors-43', 'nbehaviors-8'],
      dtype='object')
```

Summary and Next Steps

The baseline collision model described in this document can be used to measure the subsequent progress and as a reference to reproduce results. The common terminology that needs to be used to fully describe the baseline model as well as subsequent models include

- Population-configuration
- Time-window configuration
- Precise list of all metrics
- Machine learning model and hyperparameters
- Cross-validation strategy to avoid overfitting
- ROC and PR curves

The baseline collision model is a highly ideal and generic model, even though it uses a significant volume of data (all vehicles over one year across three large industries). There are many possible next steps that can be broken down into evaluation steps to better understand the current model or subsequent models, or to develop new metrics based on same or other data sources, or to develop more advanced models based on same or different data.