CAR CRASH
P R E D I C T I O N

"Modelo predictivo (Espacial) de siniestros en las calles de Santiago"

Francisca Gortari Marcelo Rovai Manuel Sacasa

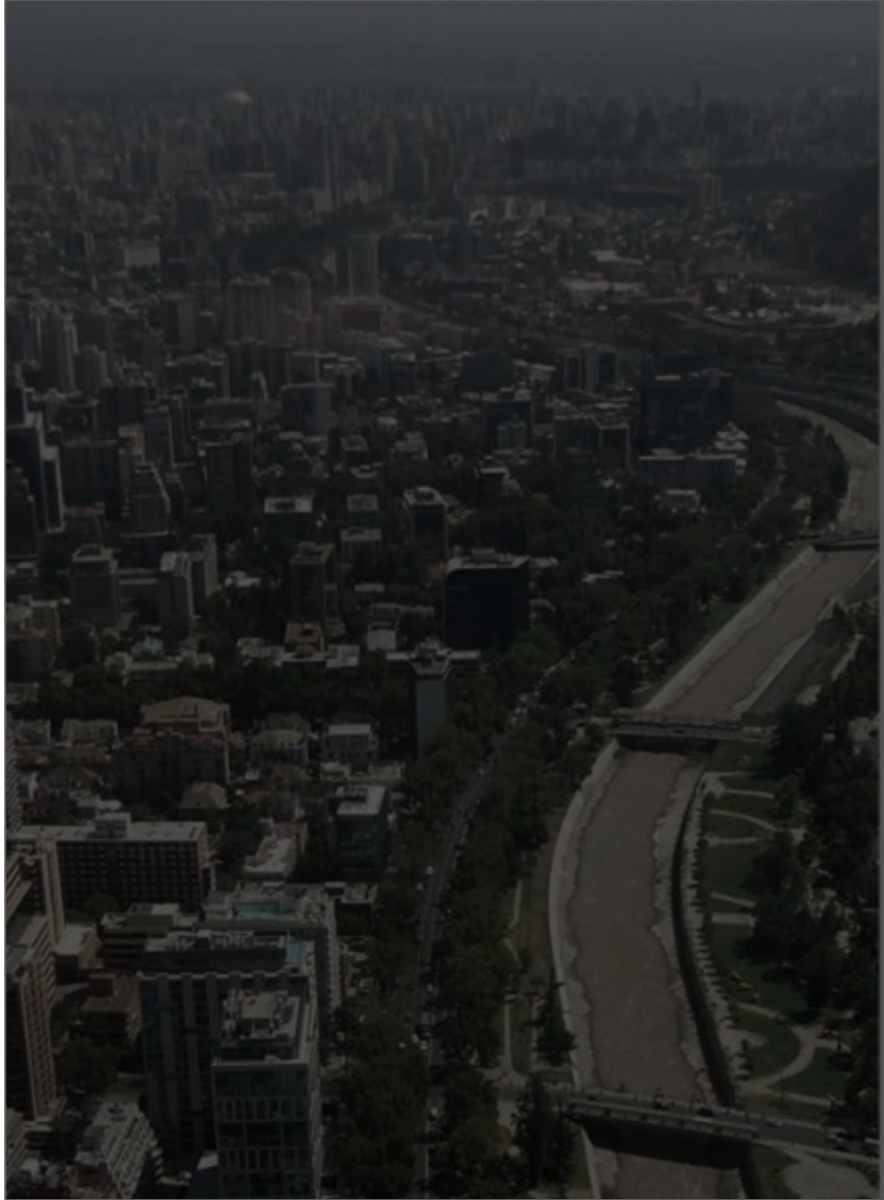
"Modelo predictivo (Espacial) de siniestros en las calles de Santiago"

UDD - Universidad del Desarrollo
MDS-18 BDA
Peredo, Oscar
Professor

Gortari, Francisca
Rovai, Marcelo
Sacasa, Manuel
Master Candidates

OBJECTIVE

The goal of the research is to predict a crash risk score for an urban grid with a 2013-2018 car crash georeferenced dataset and static urban descriptive public data set.



All the research was processed in MacBook Pro (Retina, 15-inch, 2017) with 2.9 GHz Intel Core i7 and 16GB 2133MHz LPDDR3 Memory, processing power on-premise machines.

The data science coding environment was **Anaconda's Jupyter notebooks** running Python 3.7.1 and PySpark 2.4.3. Maps, and Grids were developed mainly with GeoPandas 0.4.1, supported by several packages as Folium and OSMNX.

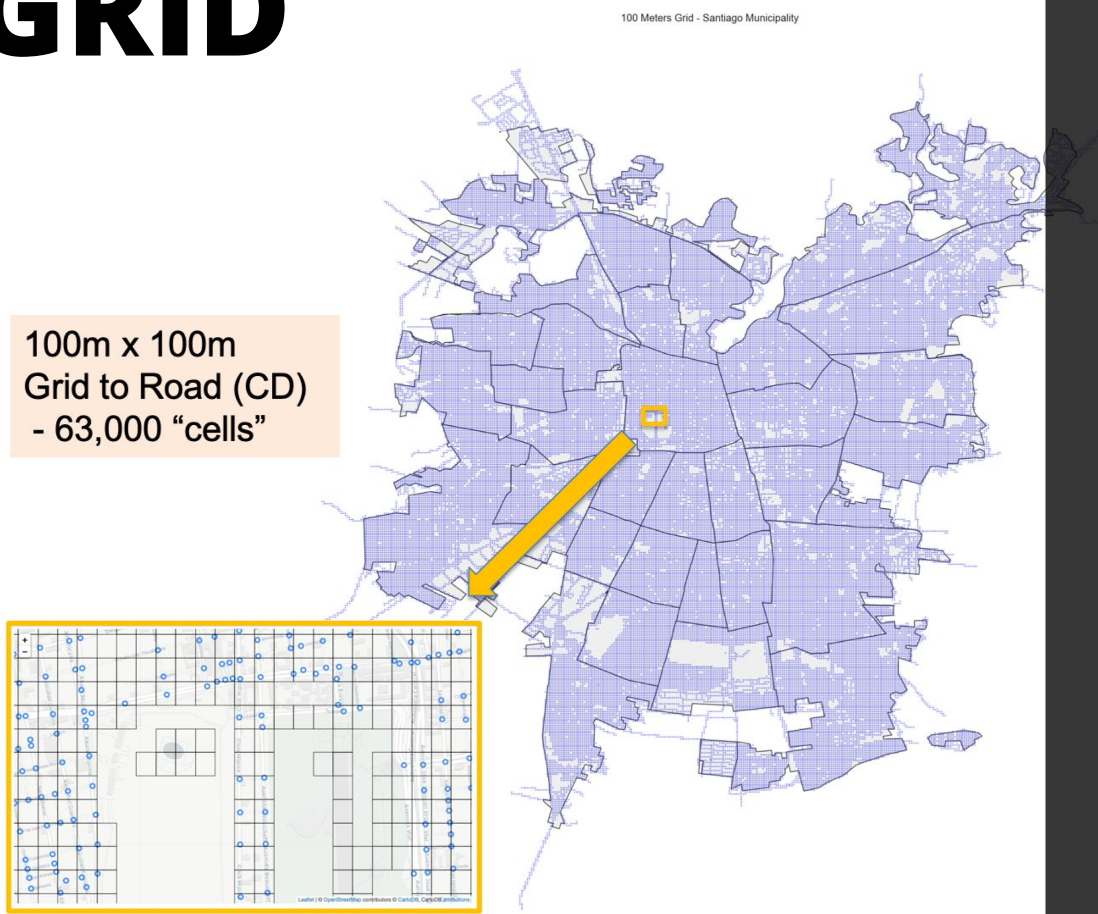
PIPELINE

Main libraries used in the end to end pipeline Jupyter Notebook

OS
Math
Numpy
Pandas
Pylab
Matplotlib
Seaborn
Folium
Networkx
Osmnx

Geopandas
Shapely
MultiLineString
Shapefile
Gpd_lite_toolbox
Findspark
Pyspark
Sparkxgb

GRID



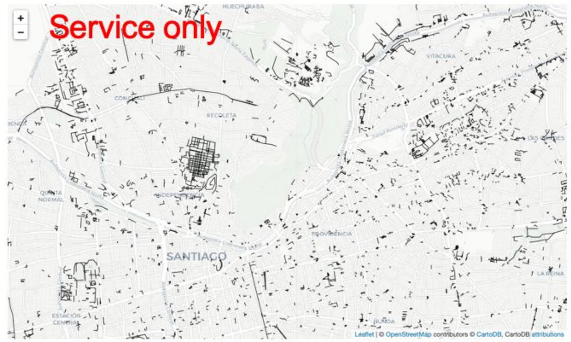
100m x 100m
Grid to Road (CD)
- 63,000 "cells"

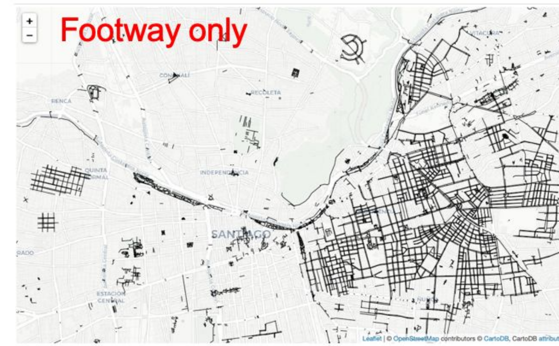
A function was created to generate grids associated to roads. The input parameters are the length of each cell and the type of roads (or streets) to be considered. For this work we used 100m and all streets except the road fclasses "Service and "Footway.

Also a 50m grid was generated for future work and it is available on grid folder.

Filtering Roads

Filtering roads is important, in order to reduce the number of "cells" to be analysed by final model, once we will work only with cells where a car accident can really happen.





Roads with Service and Footway Filter

DATA

Static Features Dataset

The static features were obtained from [OPEN STREET MAP   OSM CHILE] dataset with georeferenced points: **places, points of interest, traffic, transport, roads and intersections** (created from roads).

DATA

Static Features Dataset

Creating a Function

```python
def adding_static_feat_dist_to_grid(g,
                                    osm_shp,
                                    feature,
                                    meters,
                                    agg_type='sum'):
    pois_x = osm_shp[osm_shp.fclass == feature]
    x = pois_x[['geometry']].copy()
    dist = (meters * 0.1) / 11000
    x['geometry'] = x.geometry.buffer(dist)
    x = gpd.sjoin(g, x, how='left', op='intersects')
    x = x.fillna(0)
    feature = feature + '_' + str(meters)
    x = x.rename(columns={'index_right': feature})
    x[feature] = x[feature].apply(lambda x: 0 if x == 0.0 else 1)
    x = x.groupby('FID', as_index=False).agg({
        feature: agg_type,
        'geometry': 'first',
    })
    x = gpd.GeoDataFrame(x, crs='4326')
    print("grid shape: ", x.shape)
    print("Static Feature type {} has {} events".format(
        feature, x[feature].sum()))
    return x
```

executed in 6ms, finished 16:14:49 2019-08-02

```python
g = grid
osm_shp = pois
meters = 100
```

executed in 3ms, finished 16:14:54 2019-08-02

```python
feature = 'school'
school_100 = adding_static_feat_dist_to_grid(g, osm_shp, feature, meters)
```

executed in 6.29s, finished 16:15:06 2019-08-02

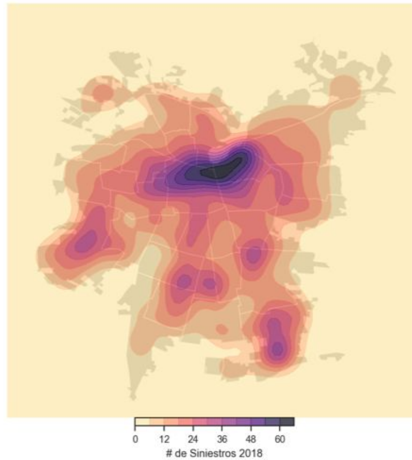
```
grid shape:  (63029, 3)
Static Feature type school_100 has 12675 events
```

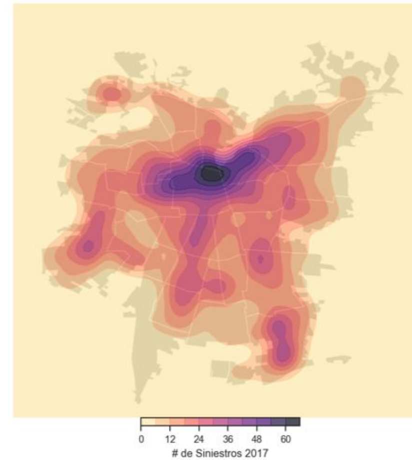
```
X                     63029 non-null float6
Y                     63029 non-null float6
bank                  63029 non-null int64
bench                 63029 non-null int64
beverages             63029 non-null int64
bus_stop              63029 non-null int64
bus_stop_100          63029 non-null int64
cafe                  63029 non-null int64
convenience           63029 non-null int64
convenience_100       63029 non-null int64
convenience_200       63029 non-null int64
crossing              63029 non-null int64
crossing_100          63029 non-null int64
fast_food             63029 non-null int64
fast_food_100         63029 non-null int64
fast_food_200         63029 non-null int64
fuel                  63029 non-null int64
intercect             63029 non-null int64
kindergarten          63029 non-null int64
motorway_junction     63029 non-null int64
parking               63029 non-null int64
parking_bicycle       63029 non-null int64
pharmacy              63029 non-null int64
railway_station       63029 non-null int64
railway_station_100   63029 non-null int64
restaurant            63029 non-null int64
restaurant_100        63029 non-null int64
school                63029 non-null int64
school_100            63029 non-null int64
school_200            63029 non-null int64
stop                  63029 non-null int64
stop_100              63029 non-null int64
taxi                  63029 non-null int64
traffic_signals       63029 non-null int64
traffic_signals_100   63029 non-null int64
turning_circle        63029 non-null int64
dtypes: float64(2), int64(34)
```

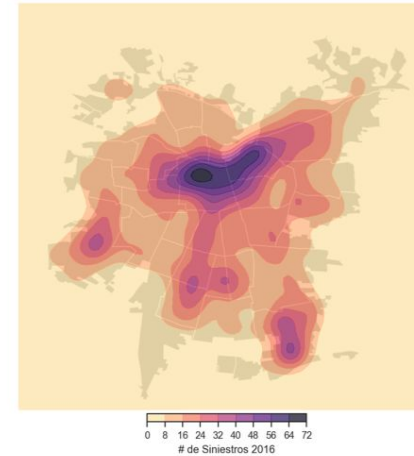
Grid individual Centroids

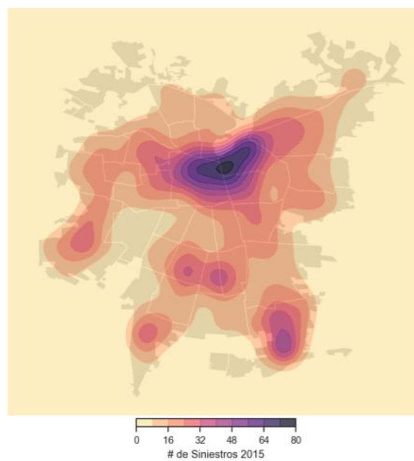
Static Features from OSM

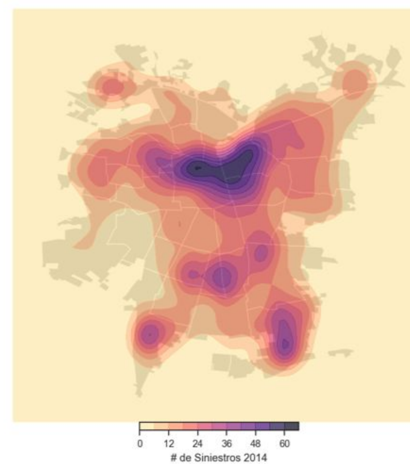
DATA
CONASET
2013-18

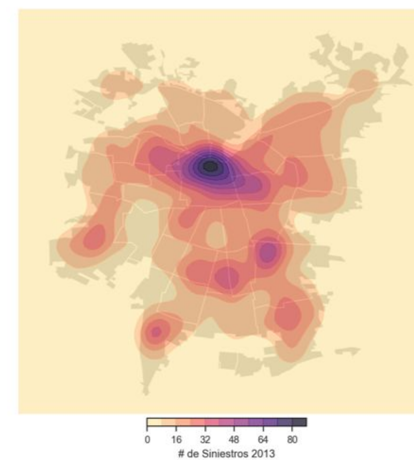












DATA

Dynamic Features Dataset

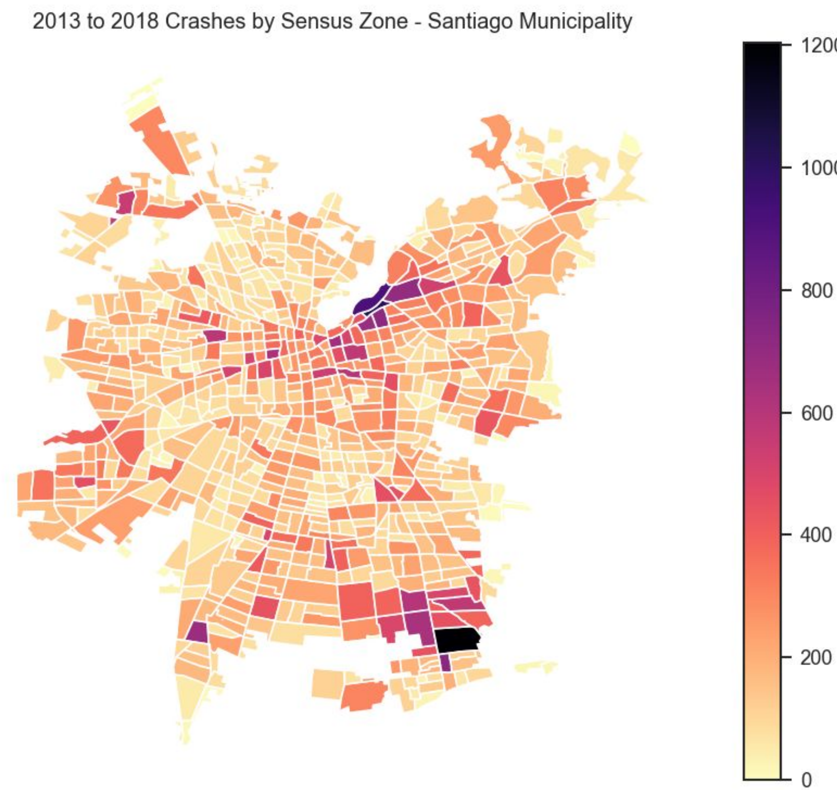
The dynamic features were obtained from CONASET

6 Datasets: Georeferenced car crashes from 2013 until 2018 enriched with injury, type of crash, wounded persons, etc. Name of the datasets **"Siniestros RM20XX".**

The "Siniestros" **datasets from 2013 to 2017 have been used to train/test the models**, while the **2018 dataset has been used to validate** the best model (Dynamic 2018 features generated from 2017 dataset - 'Year-1").

DATA

Dynamic Features Dataset



DATA

Dynamic Features Dataset

Dataset concatenated

Only Spatial data (no time)

```
siniestros.sample(10)
```

executed in 16ms, finished 17:35:27 2019-07-29

	geometry	Ano	Tipo__CONA	Fallecidos	Graves	Menos_Grav	Leves
18149	POINT (-70.57345845594295 -33.53564062681347)	2013	COLISION	0	0	0	1
56367	POINT (-70.73783930391549 -33.46285170610612)	2015	ATROPELLO	0	0	0	0
108348	POINT (-70.692421 -33.431526)	2018	COLISION	0	0	0	0
19879	POINT (-70.66134830647911 -33.55932925604031)	2013	COLISION	0	0	0	2
13145	POINT (-70.61845651655494 -33.54643059019953)	2013	COLISION	0	0	0	1
74735	POINT (-70.59810638248351 -33.4397427083309)	2016	COLISION	0	0	0	0
32657	POINT (-70.53339396042094 -33.47330588298253)	2014	ATROPELLO	0	0	0	1
90799	POINT (-70.7298924 -33.47838680000002)	2017	COLISION	0	0	0	1
6654	POINT (-70.6365284731105 -33.43756410420003)	2013	CAIDA	0	0	0	0
68232	POINT (-70.6113239 -33.4227138)	2016	CHOQUE	0	0	0	0

4.3 Adding Crash Severity

`Severity of a Crash Definition:` "The severity of a crash. Possible values are 'F' (Fallecidos), 'G' (Grave), 'M' (Menos Grave), 'L' (Leves), 'N' (non-injury). This is determined by the worst injury sustained in the crash at time of entry."

```
def sev_crash(row):
    if row['Fallecidos'] != 0: return 'F'
    elif row['Graves'] !=0: return'G'
    elif row['Menos_Grav'] !=0: return 'M'
    elif row['Leves'] !=0: return 'L'
    else: return 'N'
```

executed in 3ms, finished 17:35:51 2019-07-29

```
siniestros['SEV'] = siniestros.apply(sev_crash, axis=1)
```

executed in 3.09s, finished 17:36:06 2019-07-29

Creating a new Column Serverity Index

```
def sev_index_crash(row):
    if row['Fallecidos'] != 0: return 5
    elif row['Graves'] !=0: return 4
    elif row['Menos_Grav'] !=0: return 3
    elif row['Leves'] !=0: return 2
    else: return 1
```

executed in 4ms, finished 17:36:17 2019-07-29

```
siniestros['SEV_Index'] = siniestros.apply(sev_index_crash, axis=1)
```

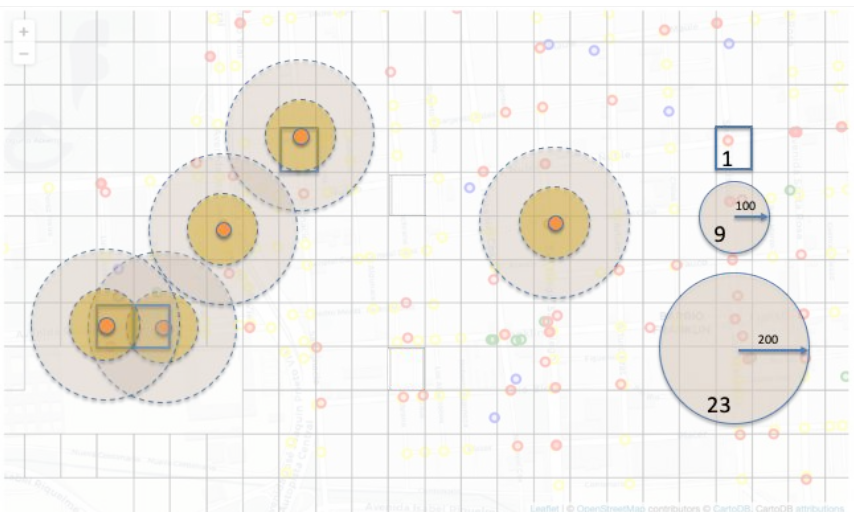
executed in 3.32s, finished 17:36:32 2019-07-29

Adding Crash Severity

DATA

Dynamic Features with events from "meters"

Example event X meters from GRID [100m]

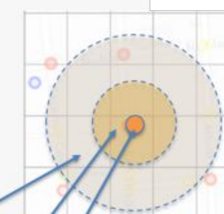


inside ➔ 2 events type orange
100m ➔ 18 events type orange
200m ➔ 46 events type orange

Function to capture event X meters from GRID

```python
def adding_sin_type_date_dist_to_grid(g, siniestros, Y, D1, D2, meters):
    s = siniestros[(siniestros.Fecha >= D1) & (siniestros.Fecha <= D2)]
    s = s[s.Tipo__CONA == Y]
    s = s[['geometry']].copy()
    dist = (meters*0.1)/11000
    s['geometry']= s.geometry.buffer(dist)
    s = gpd.sjoin(g, s, how='left', op='intersects')
    s = s.fillna(0)
    Y = Y+'_'+str(meters)
    s = s.rename(columns={'index_right': Y})
    s[Y] = s[Y].apply(lambda x: 0 if x == 0.0 else 1)
    gs = s.groupby('FID', as_index=False).agg({
        Y: 'sum',
        'geometry': 'first'
    })
    gs = gpd.GeoDataFrame(gs, crs='4326')
    print("Grid shape: ", gs.shape)
    print("Crash type {} has {} events".format(Y, gs[Y].sum()))
    return gs
```

executed in 8ms, finished 14:42:51 2019-07-26



```python
D1 = '2017-01-01'
D2 = '2017-12-31'
Y = 'INCENDIO'
meters = 200
```

executed in 3ms, finished 14:44:14 2019-07-26

```python
INCENDIO_200 = adding_sin_type_date_dist_to_grid(grid, siniestros, Y, D1, D2,
                                                  meters)
```

executed in 6.17s, finished 14:44:29 2019-07-26

```
Grid shape:  (67741, 3)
Crash type INCENDIO_200 has 128 events
```

128

```python
INCENDIO_1 = adding_sin_type_date_dist_to_grid(grid, siniestros, Y, D1, D2,
                                                meters)
```

executed in 6.13s, finished 14:49:08 2019-07-26

```
Grid shape:  (67741, 3)
Crash type INCENDIO_1 has 6 events
```

6

```python
meters = 100
```

executed in 3ms, finished 14:46:30 2019-07-26

```python
INCENDIO_100 = adding_sin_type_date_dist_to_grid(grid, siniestros, Y, D1, D2,
                                                  meters)
```

executed in 6.52s, finished 14:46:45 2019-07-26

```
Grid shape:  (67741, 3)
Crash type INCENDIO_100 has 49 events
```

49

DATA

Dynamic Features and Dependent Variable

DATA

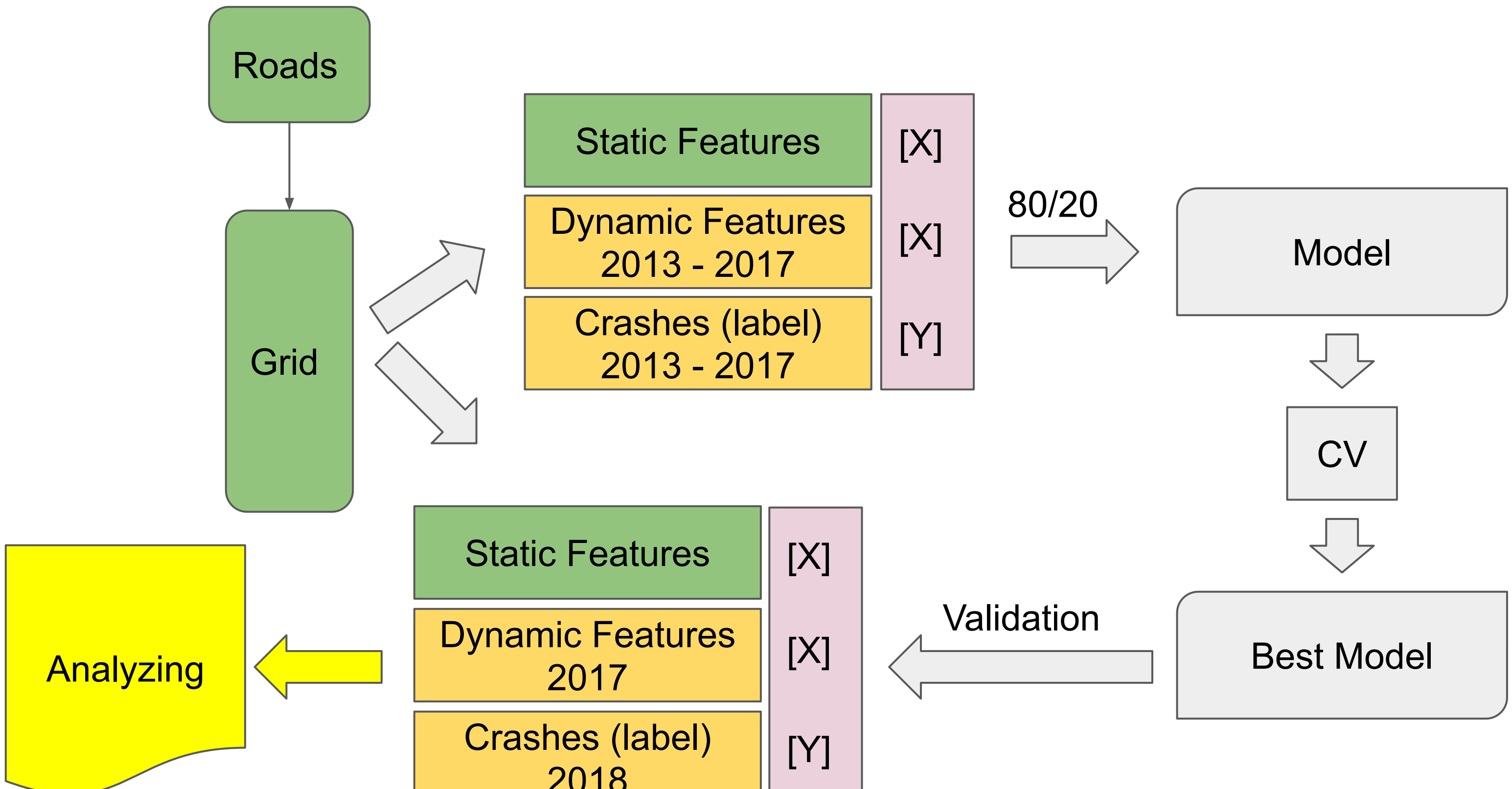
Final datasets for modelling

A dataset called **final_train_dataset_grid_100.csv,** was used for training (80%) and test (20%) models. Here, dependent variable "SINIESTRO", was generated with all real events from 2013 to 2017. Same as its dynamic features.

A dataset called **final_test_dataset_grid_100.csv,** was used for validating the best model after cross validation phase. Here, dependent variable "SINIESTRO", was generated with all real events from 2018 but its dynamic features was from 2017 only.

```
root
 |-- _c0: integer (nullable = true)
 |-- id: integer (nullable = true)
 |-- X: double (nullable = true)
 |-- Y: double (nullable = true)
 |-- bank: integer (nullable = true)
 |-- bench: integer (nullable = true)
 |-- beverages: integer (nullable = true)
 |-- bus_stop: integer (nullable = true)
 |-- bus_stop_100: integer (nullable = true)
 |-- cafe: integer (nullable = true)
 |-- convenience: integer (nullable = true)
 |-- convenience_100: integer (nullable = true)
 |-- convenience_200: integer (nullable = true)
 |-- crossing: integer (nullable = true)
 |-- crossing_100: integer (nullable = true)
 |-- fast_food: integer (nullable = true)
 |-- fast_food_100: integer (nullable = true)
 |-- fast_food_200: integer (nullable = true)
 |-- fuel: integer (nullable = true)
 |-- intercect: integer (nullable = true)
 |-- kindergarten: integer (nullable = true)
 |-- motorway_junction: integer (nullable = true)
 |-- parking: integer (nullable = true)
 |-- parking_bicycle: integer (nullable = true)
 |-- pharmacy: integer (nullable = true)
 |-- railway_station: integer (nullable = true)
 |-- railway_station_100: integer (nullable = true)
 |-- restaurant: integer (nullable = true)
 |-- restaurant_100: integer (nullable = true)
 |-- school: integer (nullable = true)
 |-- school_100: integer (nullable = true)
 |-- school_200: integer (nullable = true)
 |-- stop: integer (nullable = true)
 |-- stop_100: integer (nullable = true)
 |-- taxi: integer (nullable = true)
 |-- traffic_signals: integer (nullable = true)
 |-- traffic_signals_100: integer (nullable = true)
 |-- turning_circle: integer (nullable = true)
 |-- ATROPELLO_100: integer (nullable = true)
 |-- ATROPELLO_200: integer (nullable = true)
 |-- CAIDA_100: integer (nullable = true)
 |-- CAIDA_200: integer (nullable = true)
 |-- CHOQUE_100: integer (nullable = true)
 |-- CHOQUE_200: integer (nullable = true)
 |-- COLISION_100: integer (nullable = true)
 |-- COLISION_200: integer (nullable = true)
 |-- INCENDIO_100: integer (nullable = true)
 |-- INCENDIO_200: integer (nullable = true)
 |-- OTRO TIPO_100: integer (nullable = true)
 |-- OTRO TIPO_200: integer (nullable = true)
 |-- SEV_Index_100: double (nullable = true)
 |-- SEV_Index_200: double (nullable = true)
 |-- VOLCADURA_100: integer (nullable = true)
 |-- VOLCADURA_200: integer (nullable = true)
 |-- SINIESTRO: integer (nullable = true)
```

TASKS


Roads
Grid
Static Features
Dynamic Features
2013 - 2017
Crashes (label)
2013 - 2017
[X]
[X]
[Y]
80/20
Model
CV
Best Model
Validation
Static Features
Dynamic Features
2017
Crashes (label)
2018
[X]
[X]
[Y]
Analyzing

MODELING

Pipeline, Split data and RF modelling as a baseline

```python
categoricalColumns = cat_cols
cols = df.columns
stages = []

for categoricalCol in categoricalColumns:
    stringIndexer = StringIndexer(inputCol=categoricalCol,
                                  outputCol=categoricalCol + 'Index')
    encoder = OneHotEncoderEstimator(inputCols=[stringIndexer.getOutputCol()],
                                     outputCols=[categoricalCol + "classVec"])
    stages += [stringIndexer, encoder]

label_stringIdx = StringIndexer(inputCol='SINIESTRO', outputCol='label')
stages += [label_stringIdx]

# Assemble the columns into a feature vector
assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]
```

executed in 13ms, finished 12:14:21 2019-08-03

```python
pipeline = Pipeline(stages = stages)
pipelineModel = pipeline.fit(df)
df = pipelineModel.transform(df)
selectedCols = ['label', 'features'] + cols
df = df.select(selectedCols)
df.printSchema()
```

executed in 291ms, finished 12:14:22 2019-08-03

```python
train, test = df.randomSplit([0.8, 0.2], seed=24)
```

executed in 12ms, finished 12:14:23 2019-08-03

```python
rf = RandomForestClassifier(featuresCol='features', labelCol='label')
rfModel = rf.fit(train)
predictions = rfModel.transform(test)
```

executed in 2.70s, finished 12:14:27 2019-08-03

```python
predictions.select('id', 'label', 'rawPrediction', 'prediction',
                   'probability').show(10)
```

executed in 477ms, finished 12:14:30 2019-08-03

```
+---+-----+--------------------+----------+--------------------+
| id|label|       rawPrediction|prediction|         probability|
+---+-----+--------------------+----------+--------------------+
|  0|  0.0|[19.3985287329805...|       0.0|[0.96992643664902...|
|  3|  0.0|[19.3985287329805...|       0.0|[0.96992643664902...|
| 14|  0.0|[19.3985287329805...|       0.0|[0.96992643664902...|
| 18|  0.0|[19.3985287329805...|       0.0|[0.96992643664902...|
| 23|  0.0|[19.3985287329805...|       0.0|[0.96992643664902...|
| 33|  0.0|[19.3985287329805...|       0.0|[0.96992643664902...|
| 34|  0.0|[19.3985287329805...|       0.0|[0.96992643664902...|
| 44|  0.0|[19.3985287329805...|       0.0|[0.96992643664902...|
| 56|  0.0|[19.3985287329805...|       0.0|[0.96992643664902...|
| 54|  0.0|[19.3985287329805...|       0.0|[0.96992643664902...|
+---+-----+--------------------+----------+--------------------+
only showing top 10 rows
```

```python
evaluator = BinaryClassificationEvaluator()
print("Test Area Under ROC: " + str(
    evaluator.evaluate(predictions, {evaluator.metricName: "areaUnderROC"})))
```

executed in 617ms, finished 12:14:43 2019-08-03

```
Test Area Under ROC: 0.8294667464568877
```

```python
evaluator = MulticlassClassificationEvaluator()
accuracy = evaluator.evaluate(predictions, {evaluator.metricName: "accuracy"})
print("Accuracy: " + str(accuracy))
```

executed in 798ms, finished 12:14:57 2019-08-03

```
Accuracy: 0.7782577959311916
```

```python
print("Test Error = %g" % (1.0 - accuracy))
```

executed in 3ms, finished 12:15:01 2019-08-03

```
Test Error = 0.221742
```

MODELING

GBT modelling and Feature Importance

idx	name	score
42	COLISION_100	0.439067
11	crossing	0.090049
17	intercect	0.071686
5	bus_stop	0.055971
33	traffic_signals	0.047034
36	ATROPELLO_100	0.035349
1	Y	0.028571
6	bus_stop_100	0.023284
34	traffic_signals_100	0.021061
48	SEV_Index_100	0.016812
26	restaurant_100	0.016779
12	crossing_100	0.015766
25	restaurant	0.015591
46	OTRO TIPO_100	0.014711
50	VOLCADURA_100	0.013828
22	pharmacy	0.013157
43	COLISION_200	0.012653
0	X	0.011431
41	CHOQUE_200	0.009117
38	CAIDA_100	0.008853

```
gbt = GBTClassifier(maxIter=10)
```
executed in 7ms, finished 12:15:07 2019-08-03

```
gbtModel = gbt.fit(train)
```
executed in 7.27s, finished 12:15:15 2019-08-03

```
predictions = gbtModel.transform(test)
```
executed in 48ms, finished 12:15:17 2019-08-03

```
predictions.select('id', 'label', 'rawPrediction', 'prediction',
                   'probability').show(5)
```
executed in 454ms, finished 12:15:19 2019-08-03

```
+---+-----+--------------------+----------+--------------------+
| id|label|       rawPrediction|prediction|         probability|
+---+-----+--------------------+----------+--------------------+
|  0|  0.0|[1.32412711336971...|       0.0|[0.93390330915520...|
|  3|  0.0|[1.32412711336971...|       0.0|[0.93390330915520...|
| 14|  0.0|[1.32412711336971...|       0.0|[0.93390330915520...|
| 18|  0.0|[1.32412711336971...|       0.0|[0.93390330915520...|
| 23|  0.0|[1.32412711336971...|       0.0|[0.93390330915520...|
+---+-----+--------------------+----------+--------------------+
only showing top 5 rows
```

```
evaluator = BinaryClassificationEvaluator()
print("Test Area Under ROC: " + str(
    evaluator.evaluate(predictions,
                       {evaluator.metricName: "areaUnderROC"})))
```
executed in 542ms, finished 12:15:22 2019-08-03

```
Test Area Under ROC: 0.8383690618564904
```

```
evaluator = MulticlassClassificationEvaluator()
accuracy = evaluator.evaluate(predictions, {evaluator.metricName: "accuracy"})
print("Accuracy: " + str(accuracy))
```
executed in 720ms, finished 12:15:29 2019-08-03

```
Accuracy: 0.7879978006440971
```

```
print("Test Error = %g" % (1.0 - accuracy))
```
executed in 4ms, finished 12:15:30 2019-08-03

```
Test Error = 0.212002
```

MODELING

XGBoost

```
xgboost = XGBoostEstimator(
    featuresCol="features",
    labelCol="label",
    predictionCol="prediction"
)
```

executed in 58ms, finished 13:35:42 2019-08-03

```
xgbModel = xgboost.fit(train)
```

executed in 2.83s, finished 13:35:48 2019-08-03

```
predictions = xgbModel.transform(test)
```

executed in 298ms, finished 13:35:49 2019-08-03

```
predictions.select('id', 'label', 'prediction').show(5)
```

executed in 806ms, finished 13:35:50 2019-08-03

```
+---+-----+----------+
| id|label|prediction|
+---+-----+----------+
|  0|  0.0|       0.0|
|  3|  0.0|       0.0|
| 14|  0.0|       0.0|
| 18|  0.0|       0.0|
| 23|  0.0|       0.0|
+---+-----+----------+
only showing top 5 rows
```

```
evaluator = MulticlassClassificationEvaluator(labelCol='label', metricName="accuracy")
```

executed in 9ms, finished 13:35:54 2019-08-03

```
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g" % (1.0 - accuracy))
```

executed in 1.79s, finished 13:35:57 2019-08-03

```
Test Error = 0.210117
```

```
Almost same as Test Error (0.212002) got with GBT Default parameters.
```

MODELING

Tuning and saving best GBT Model

```
predictions.select('id', 'label', 'rawPrediction', 'probability', 'prediction').show(10)
```

executed in 502ms, finished 12:26:23 2019-08-03

```
+---+-----+--------------------+--------------------+----------+
| id|label|       rawPrediction|         probability|prediction|
+---+-----+--------------------+--------------------+----------+
|  0|  0.0|[1.54341621871724...|[0.95634631650520...|       0.0|
|  3|  0.0|[1.54341621871724...|[0.95634631650520...|       0.0|
| 14|  0.0|[1.54341621871724...|[0.95634631650520...|       0.0|
| 18|  0.0|[1.54341621871724...|[0.95634631650520...|       0.0|
| 23|  0.0|[1.54341621871724...|[0.95634631650520...|       0.0|
| 33|  0.0|[1.54341621871724...|[0.95634631650520...|       0.0|
| 34|  0.0|[1.54341621871724...|[0.95634631650520...|       0.0|
| 44|  0.0|[1.54341621871724...|[0.95634631650520...|       0.0|
| 56|  0.0|[1.54341621871724...|[0.95634631650520...|       0.0|
| 54|  0.0|[1.54341621871724...|[0.95634631650520...|       0.0|
+---+-----+--------------------+--------------------+----------+
only showing top 10 rows
```

```
bestModel = cvModel.bestModel
```

executed in 3ms, finished 12:26:25 2019-08-03

```
bestModel
```

executed in 5ms, finished 12:26:26 2019-08-03

```
GBTClassificationModel (uid=GBTClassifier_95ac8c178565) with 20 trees
```

```
bestModel.write().overwrite().save('../model/GeoProjectBestModel_1.model')
```

executed in 721ms, finished 12:26:28 2019-08-03

```
paramGrid = (ParamGridBuilder()
             .addGrid(gbt.maxDepth, [4, 6])
             .addGrid(gbt.maxBins, [40, 60, 70])
             .addGrid(gbt.maxIter, [10, 20])
             .build())

cv = CrossValidator(estimator=gbt,
                    estimatorParamMaps=paramGrid,
                    evaluator=evaluator,
                    numFolds=5)

# Run cross validations.  This can take about 7.3 minutes!
cvModel = cv.fit(train)
predictions = cvModel.transform(test)
```

executed in 7m 15s, finished 12:23:56 2019-08-03

```
evaluator = BinaryClassificationEvaluator()
print("Test Area Under ROC: " + str(
    evaluator.evaluate(predictions,
                       {evaluator.metricName: "areaUnderROC"})))
```

executed in 537ms, finished 12:25:40 2019-08-03

```
Test Area Under ROC: 0.841362042678081
```

```
evaluator = MulticlassClassificationEvaluator()
accuracy = evaluator.evaluate(predictions, {evaluator.metricName: "accuracy"})
print("Accuracy: " + str(accuracy))
```

executed in 766ms, finished 12:25:42 2019-08-03

```
Accuracy: 0.7924750608750295
```

```
print("Test Error = %g" % (1.0 - accuracy))
```

executed in 3ms, finished 12:25:45 2019-08-03

```
Test Error = 0.207525
```

VALIDATING

Best GBT Model with 2018 dataset

```
valModel = GBTClassificationModel.load("../model/GeoProjectBestModel_1.model")
```

executed in 455ms, finished 12:39:11 2019-08-03

Load Best Model

Loading the dataset with 2018 data:

```
df_2018 = spark.read.csv('../data/final_test_dataset_grid_100.csv',
                         header=True,
                         inferSchema=True)
df_2018.printSchema()
```

executed in 308ms, finished 12:39:13 2019-08-03

Load Data

```
root
 |-- _c0: integer (nullable = true)
 |-- id: integer (nullable = true)
 |-- X: double (nullable = true)
 |-- Y: double (nullable = true)
```

```
num_cols, cat_cols = find_num_cat_features(df_2018)
```

executed in 4ms, finished 12:39:13 2019-08-03

```
0  categorical features
54  numerical features
```

```
df_2018.groupby('SINIESTRO').count().toPandas()
```

executed in 216ms, finished 12:39:14 2019-08-03

	SINIESTRO	count
0	1	6687
1	0	56342

Note that on this dataset, we have around 10.5% of the Grid's cells with crash events and 89.4% with No events.

```
numericCols = num_cols[1:-1] # Taking out id and Target variable
numericCols
```

executed in 5ms, finished 12:39:18 2019-08-03

```
['X',
 'Y',
 'bank',
 'bench',
```

```
categoricalColumns = cat_cols
cols = df.columns
stages = []

for categoricalCol in categoricalColumns:
    stringIndexer = StringIndexer(inputCol=categoricalCol,
                                  outputCol=categoricalCol + 'Index')
    encoder = OneHotEncoderEstimator(inputCols=[stringIndexer.getOutputCol()],
                                     outputCols=[categoricalCol + "classVec"])
    stages += [stringIndexer, encoder]

label_stringIdx = StringIndexer(inputCol='SINIESTRO', outputCol='label')
stages += [label_stringIdx]

assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]

pipeline = Pipeline(stages = stages)
pipelineModel = pipeline.fit(df_2018)
df_2018 = pipelineModel.transform(df_2018)
selectedCols = ['label', 'features'] + cols
df_2018 = df_2018.select(selectedCols)
df_2018.printSchema()
```

executed in 242ms, finished 12:39:19 2019-08-03

Prepare Data

```
evaluator = BinaryClassificationEvaluator()
print("Test Area Under ROC: " + str(
    evaluator.evaluate(predict_2018,
                       {evaluator.metricName: "areaUnderROC"})))
```

Apply Best Model

executed in 994ms, finished 12:42:35 2019-08-03

```
Test Area Under ROC: 0.7742939707280319
```

```
evaluator = MulticlassClassificationEvaluator()
accuracy = evaluator.evaluate(predict_2018, {evaluator.metricName: "accuracy"})
print("Accuracy: " + str(accuracy))
```

executed in 981ms, finished 12:42:48 2019-08-03

```
Accuracy: 0.8942708911770773
```

```
print("Test Error = %g" % (1.0 - accuracy))
```

executed in 3ms, finished 12:40:51 2019-08-03

```
Test Error = 0.105729
```

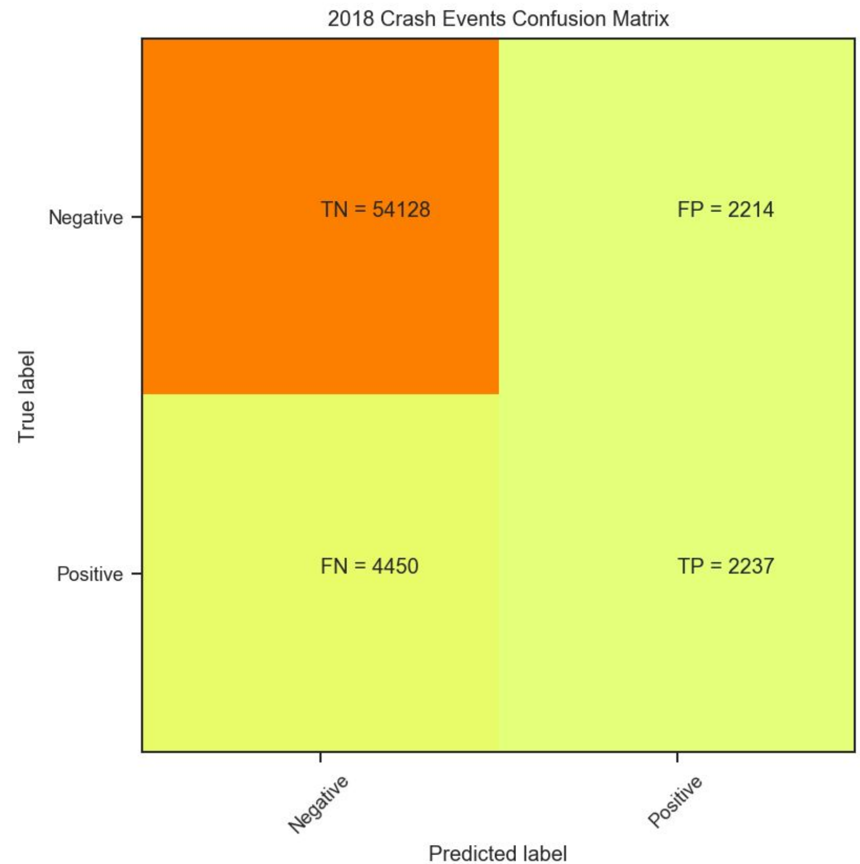
ANALYZING

Results from Best Model

```
1  print(classification_report(y_test,y_pred))
```

executed in 133ms, finished 13:14:48 2019-08-03

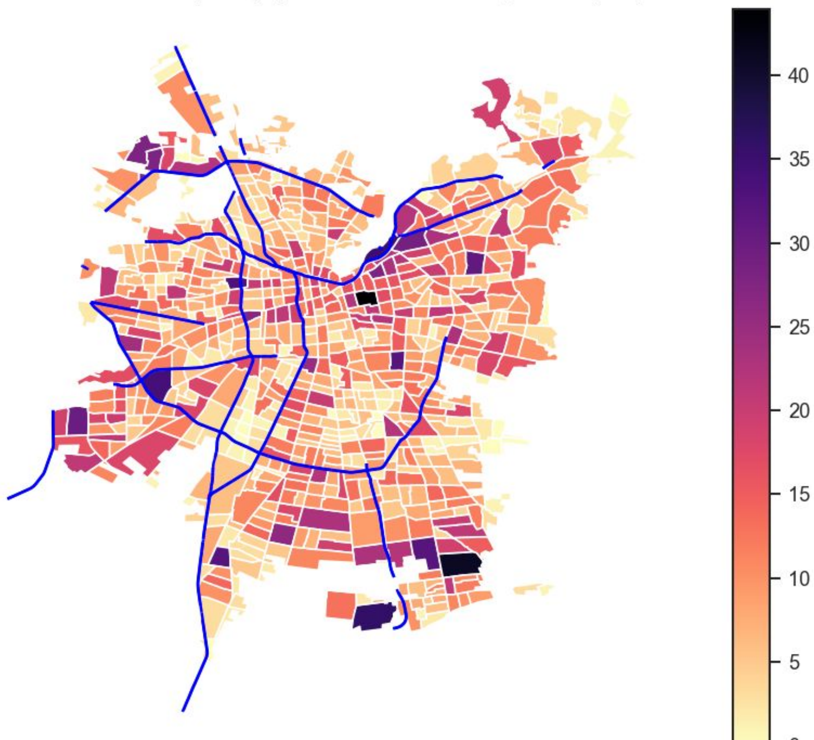
	precision	recall	f1-score	support
0.0	0.92	0.96	0.94	56342
1.0	0.50	0.33	0.40	6687
accuracy			0.89	63029
macro avg	0.71	0.65	0.67	63029
weighted avg	0.88	0.89	0.88	63029



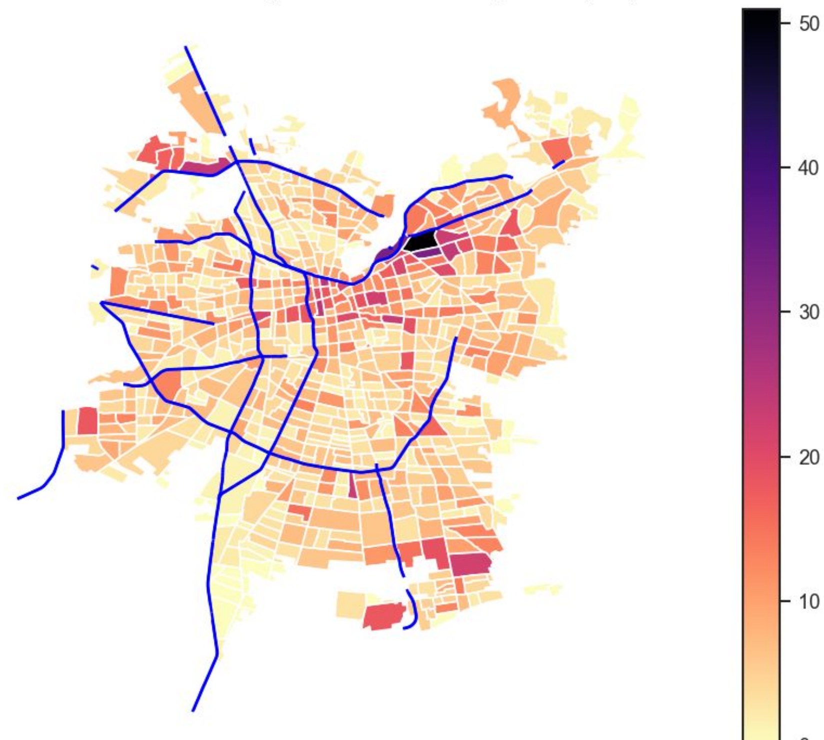
ANALYZING

Label versus Prediction

2018 Real Crashes (Label) by Sensus Zone - Santiago Municipality

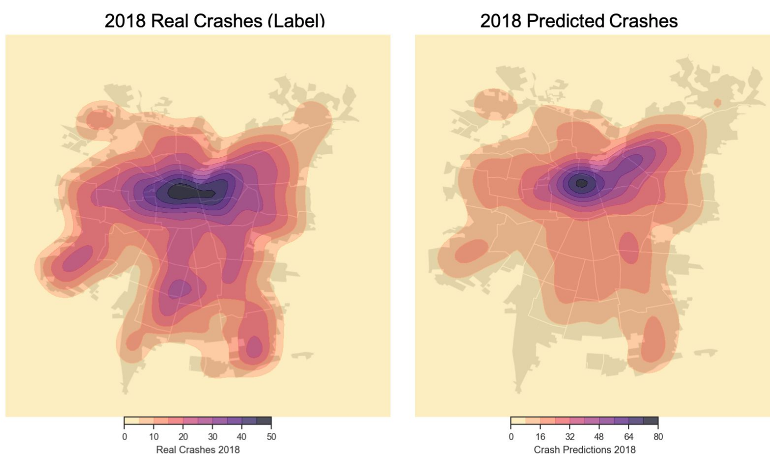


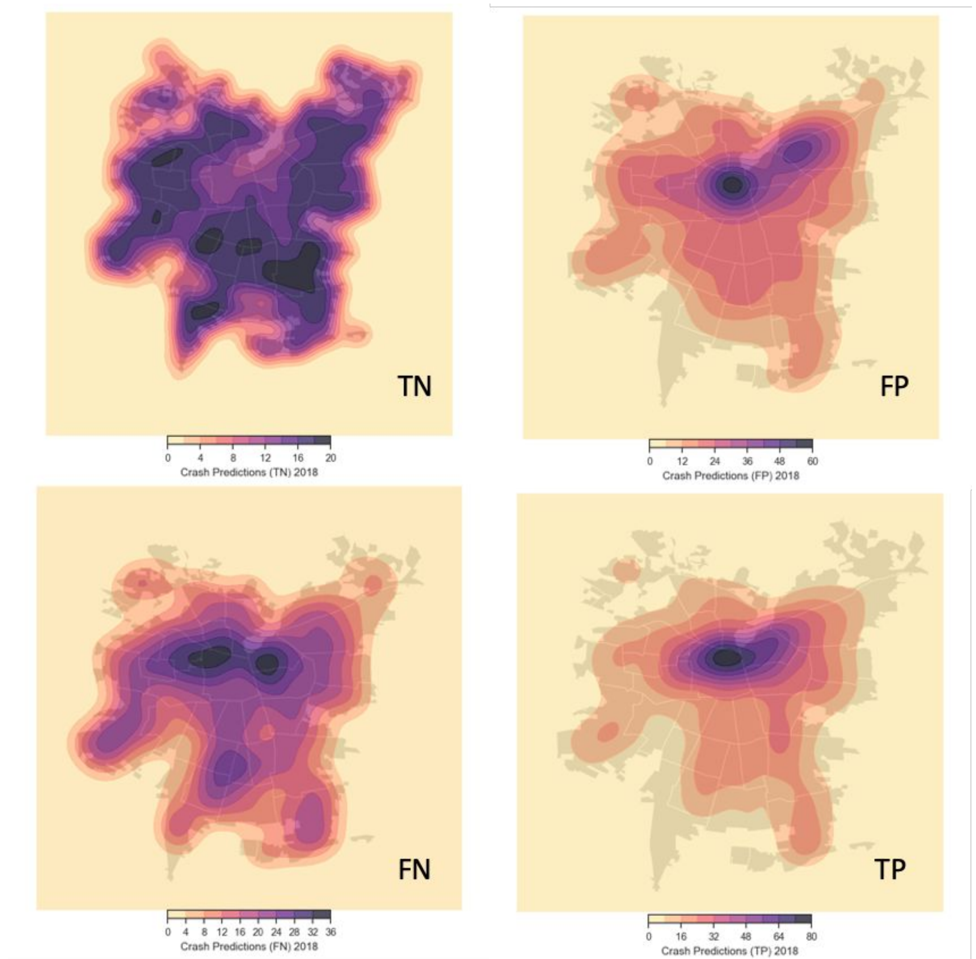
2018 Crash Predictions by Sensus Zone - Santiago Municipality



ANALYZING

Label versus Prediction





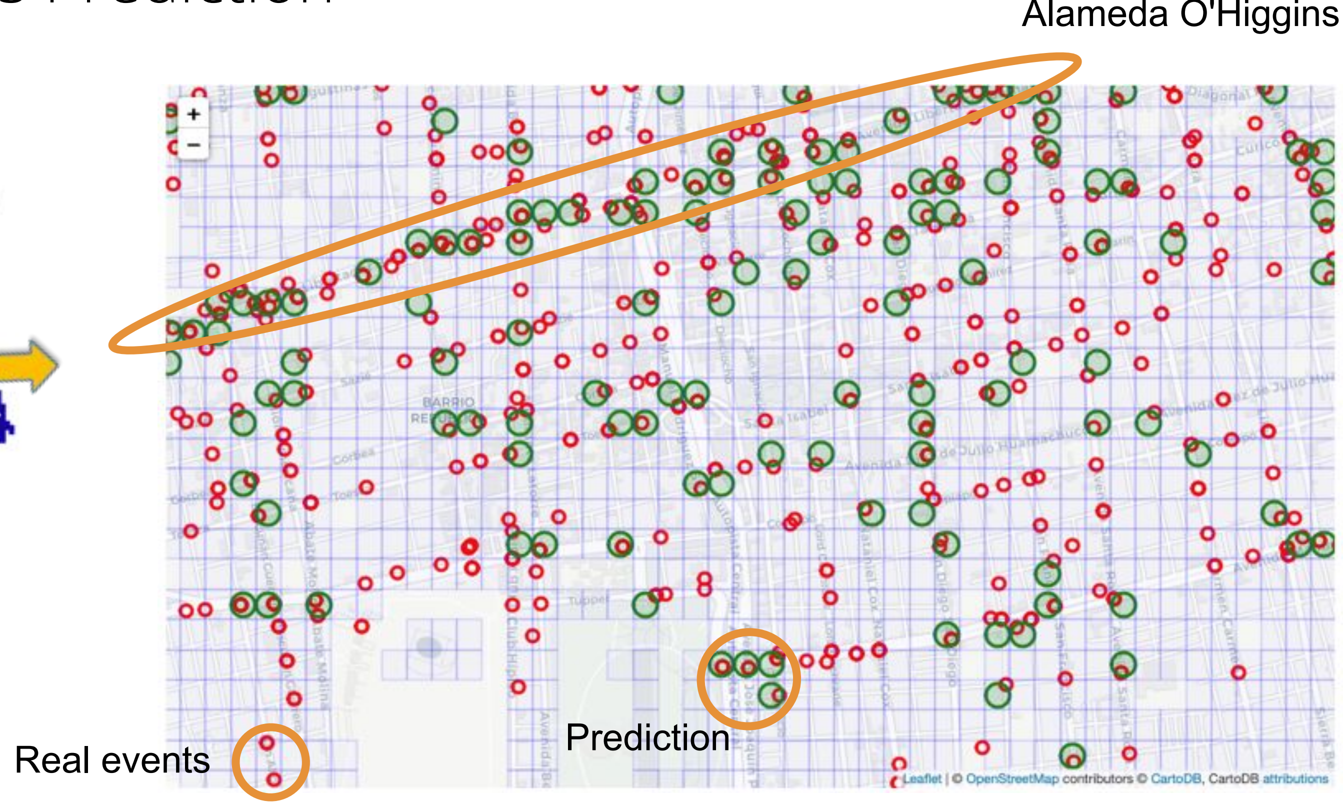
Proximity with highways/main roads seems to be an important factor of events (not well captured by model)

ANALYZING

Label versus Main Roads

ANALYZING

Label versus Prediction

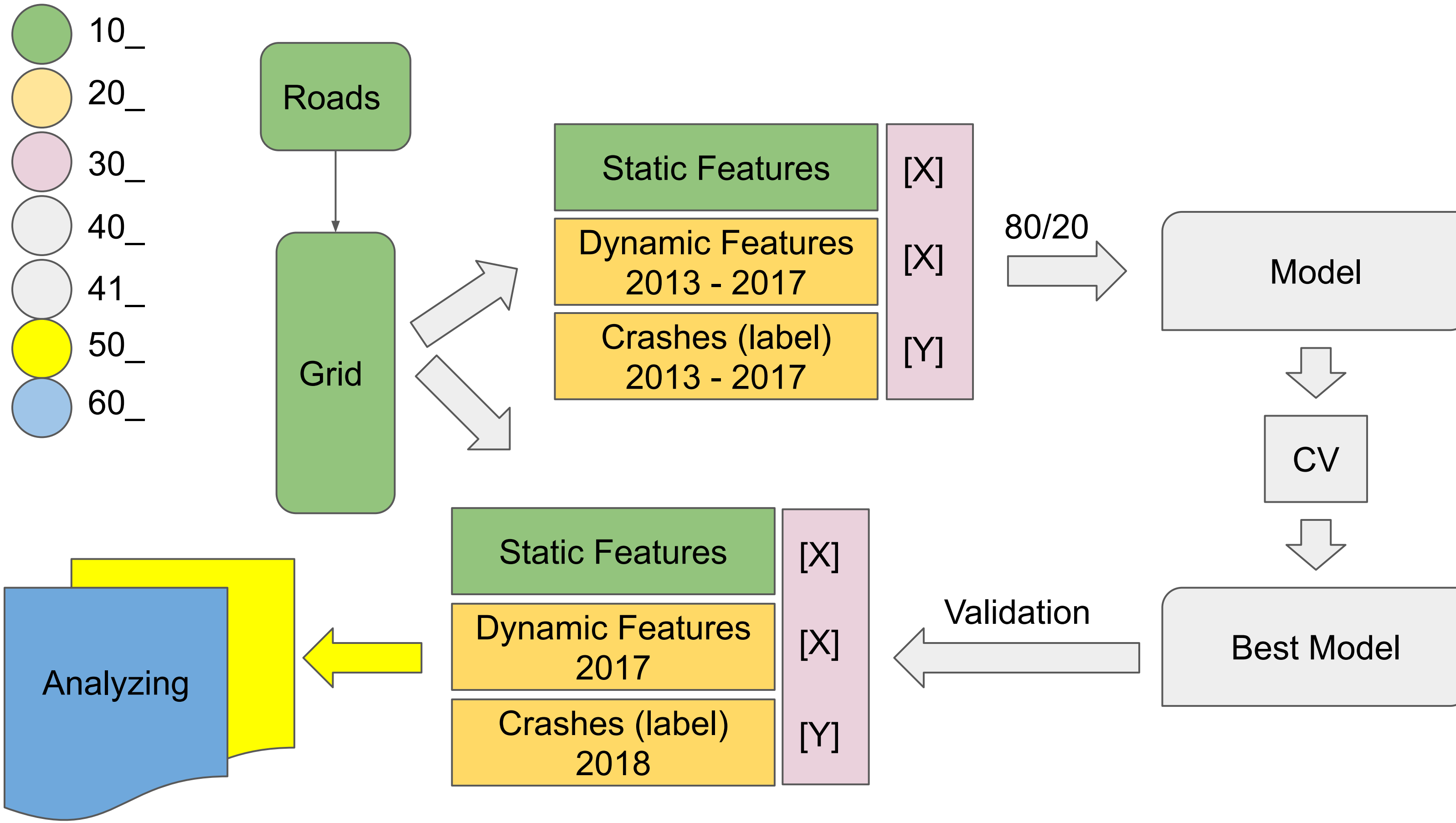


FUTURE WORK

- Test model with different grids (50m and 200m for example)
- Form a dataset only with years that have more data, adding temporal dynamic features as working days, street conditions, clima, etc.
- Introduce new static features as:
    - Average road speed
    - Demographic density
    - Mobility factor
    - Roads geometry (curvature, inclination, etc.)
    - Altitude
    - Number of road lanes
    - Proximity with highways/main roads
    - Other

Addendum

NOTEBOOKS KEY



NOTEBOOKS CONTENT

10_

1 Datasets (Landing)
- 1.1 CONASET
- 1.2 OpenStreetMap

2 Main Libraries

3 Main Functions

4 Getting Raw Data
- 4.1 Restrict Analysis inside Santiago Municipality Area
- 4.2 Getting POIs
- 4.3 Getting Roads
- 4.4 Getting Traffic Points
- 4.5 Getting Transportation Points
- 4.6 Getting Roads Intersections
- 4.7 Visualizing a sample of Intersections (Santiago County)

5 Creating a grid
- 5.1 Grid related with Intersections
- 5.2 Creating a generic grid creation function
- 5.3 Testing generic Grid
- 5.4 Grid related to Roads (Calles Discretizadas 'CD')
- 5.5 Creating a 50 meters Grid to Roads
- 5.6 Testing Grid 100 to roads
- 5.7 Testing Grid 50 to roads

6 Adding Static features to road grid ('CD')
- 6.1 Including Intersections on Grid
- 6.2 Join POI & Grid
- 6.3 Join Traffic Points to Grid
- 6.4 Join Transportation Points to Grid

7 Adding new Static Features on a distance (meters)
- 7.1 Creating a Function

8 Creating a Static Features dataset
- 8.1 Creating a .CSV Static Feature dataset

20_

1 Datasets (Landing)
- 1.1 CONASET
- 1.2 OpenStreetMap

2 Main Libraries

3 Main Functions

4 Getting Raw Data
- 4.1 Loading data from CONASET
- 4.2 Restrict Siniestros inside Santiago Municipality Area
- 4.3 Adding Crash Severity

5 Filter crash type per year

6 Adding Crash related features to grid
- 6.1 Defining a generic Function to add a crash event by type and date range to the grid

7 Define Features from 2013 to 2017 to be used as Train
- 7.1 Defining Crash Type Buffer
- 7.2 Defining a generic Function to add a crash event by type, distance in meters and date range to the grid
- 7.3 Defining a generic Function to add a crash dinamic features and date range to the grid

8 Creating a Train dataset
- 8.1 Creating a .CSV Static Feature dataset

9 Defining Features from 2018 to be used as a Test dataset
- 9.1 Crash Events by type
- 9.2 Crash Events per type and distance

10 Creating a Test dataset
- 10.1 Creating a .CSV Static Feature dataset

NOTEBOOKS CONTENT

30_

1 Datasets (Landing)
  1.1 CONASET
  1.2 OpenStreetMap
2 Main Libraries
3 Creating Final Train and Test datasets
4 Creating and Saving Train Dataset
5 Creating and Saving Test Dataset

50_

1 Datasets (Landing)
  1.1 CONASET
  1.2 OpenStreetMap
2 Main Libraries
3 Analyzing the Crash Prevision 2018 Dataset

40_

1 Preliminar Installation
2 General functions
3 Libraries and pySpark initialization
4 Dataset - Generated from Notebook:
5 Verifying dataset balance
6 Preparing Data for Machine Learning
7 RF Model
8 GBT Model
9 Tuning The GBT Model
10 Validating Best Model with 2018 dataset

60_

1 Datasets (Landing)
  1.1 CONASET
  1.2 OpenStreetMap
2 Main Libraries and Initialization
3 Main Functions
4 Import and visualize dataset
5 Model Result Spatial Visualization
6 Test Results on Santiago County
7 Working with "Sensal Zones"
  7.1 Create a map with Crashes and zones

41_

1 Preliminar Installations
2 General functions
3 Libraries and pySpark initialization
4 Dataset - Generated from Notebook:
5 Verifying dataset balance
6 Preparing Data for Machine Learning
7 XGBoost Model

CAR CRASH

P R E D I C T I O N

"Modelo predictivo (Espacial) de siniestros en las calles de Santiago"

Francisca Gortari Marcelo Rovai Manuel Sacasa