Mathematical Institute

Internship Report

Statistical Science for the Life and Behavioural Sciences

Internship Report
A Bird Mass Prediction Model for Schiphol Airport

Author:
Bingjing Gu

Internal Supervisor:
Dr. T.A.L. van Erven
Mathematical Institute

External Supervisor:
Diederik Meijerink
Rok Mihevc
Data Innovation Lab Schiphol Group

Aug 2016

Contents

Chapter 1

Background on the Data Innovation Lab

Schiphol Group is an enterprise that operates airports in the Netherlands, conducting international activities and participates in airports abroad. Operating Amsterdam Airport Schiphol is the Group's largest activity.[1] *Schiphol Group*'s ambition is to continue to develop Amsterdam Airport Schiphol into Europe's preferred airport for travelers and airlines alike and to become the best data driven digital airport in the world. Data science is the key to fulfilling this ambition. The Data Innovation Lab, which is part of the Business & Technology Center of the IT department of *Schiphol Group*, aims to take maximum advantage of data to increase the business value of Amsterdam Airport Schiphol through consolidating all its data in a scalable big data infrastructure. The data infrastructure is set up and managed by *GoDataDriven.* All data from *Schiphol Group* are consolidated in the Cloudera big data solution on Microsofts cloud hosting platform Azure which is a ready to use platform with all the necessary tools for data scientists, such as R,RStudio,Python and Jupyter notebook, By applying machine learning, deep learning, statistical modeling and other advanced analytics, the Data Innovation Lab would extract more value out of internal and external data sources and deliver new trends, insights and patterns from it. To improve operational efficiency, Data Innovation Lab has already worked on the assets project, like conveyor belts and elevators, security project, parking projects, capacity project and so on. I started my internship at the Data Innovation Lab from February and was involved in developing a predictive model to solve a so called 'Angry Birds' project during this six month internship.

Chapter 2

Prediction Project

2.1 Background

Birds can be a genuine risk for aircraft in the vicinity of the Amsterdam Airport Schiphol. To ensure the highest possible degree of aviation safety, Schiphol takes measures to keep the birds as far away from the aircraft as possible. Schiphol employs bird controllers only for this purpose to keep the birds away at all hours of the night and day. However, bird control is currently done based on expertise and intuition by manual observation and from radar images provided by *Robin Radar*. Bird control might therefore be suboptimal since it is not a data driven operation. The bird control team expect us to make a predictive model to avoid purposeless chase and waiting. They prefer to know where they should go in the next several minutes to take measures on those high risk birds.

2.2 Preliminaries

In the beginning, we set up a meeting among bird control team, *Robin Radar* who is the data vendor on the bird data and us to thoroughly understand the demands of the bird control team and the meaning and structure of the data from *Robin Radar*. *Robin Radar* tracking system describes the flight path and corresponding information of an object or a group of objects detected by radar. It is a good idea to predict the mass density or location point count density in grid since the trajectory of birds shows they only fly through a short distance and short time period while the bird control team still needs enough time to drive to the scene which means per bird prediction is not practical in this case. In addition, the bird control team did not provide clear predictive target. Therefore I start from predicting the numbers of location points and then focus on the average mass in a certain area. Before starting to build the predictive model, I should think about the idea of model structure and manipulate the data first. According to the first law of geography, all things are related, but nearby things are more related than distant things[2]. My idea is to make good use of spatial and non-spatial features to predict the mass and location point numbers in the next several minutes. The runway area is extended by calculating the maximum distance from two coordinates on spherical earth which the birds may fly through and then divide into a 6 by 4 grid according to the demand and suggestion from bird control team. After that, the features were aggregated by time and cells. Considering the characteristics of birds moving, I set five minutes as rolling windows which means the predictors was estimated by the previous five minutes features. I believe the weather condition would also have influence on the predictors more or less. The weather data collected within every 10 minutes which was provided by air traffic control was also joined into the bird data as well. Every grid cells share the same weather condition at the same moment. There are 468 features and 357080 observations(8 cells in 31 days with 1440 minutes per day excluding the first five minutes) in one month data after data preparation. The spatial features include the mean value of velocity, airspeed, heading, vertical heading, peak mass, mass, mass correction, coordinates and position count. Other features have not spatial properties, which means they share the same value in different area, such as hours of the time, time slot of the day, visibility, cloud layer for coverage, average forecasted winddirection, average forecasted windspeed, average forecasted windspeed including gust and forecasted showers of rain, snow, thunderstorms.

2.3 Methods & Application

Denoting the mean mass of the specific area to be modelled by $y(s,i)$ referring to area location s and time i, we have the spatial temporal model $y(s,i) = \mu(s,i) + \nu(i) + \epsilon(s,i)$,where $y(s,i)$ denotes the spatio-temporal observations(mean mass here), $\mu(s,i)$ is the structured spatial effect, $\nu(i)$ is the non-spatial effect and $\epsilon(s,i)$ is the residual. The spatial effect is modelled as: $\mu(s,i) = \sum_{j=1}^{m} \beta_j \phi_j(s,i)$ where $\phi(s,i)$ are spatial covariates and β_j are coefficients for the spatial covariates. The non- spatial effect is modelled as: $\nu(i) = \sum_{j=1}^{q} \gamma_j \varphi_j(i)$ where $\varphi(i)$ are non-spatial covariates and γ_j are coefficients for the non-spatial covariates. Since I already melt the spatial effect into each spatial covariates during the data manipulation which would provide temporal evolution at each spatial location. In particular, we derive a one-row expression for each spatial effect and specify which area location they belong to. For instance, the spatial effect velocity at some specific area would be derived into an area indicator and top-left, top-middle, top-right, left, middle, right, bottom-left, bottom-middle, bottom-right velocity values instead of a spatial correlation indicator matrix. This procedure would be applied into each spatial effect. Moreover, we don't introduce the interaction effect between the cells and spatial effect here and only estimate the difference among the cells by the feature of cells indicator. When combining these with non-spatial features together,we could have the final model as : $y(s,i) = y_i(s) = \sum_{j=1}^{p} \beta_j x_{ij}(s) + \beta_0 + \epsilon$ where $y_i(s)$ represents the mean mass value at time i for a given cell s, $x_{ij}(s)$ include spatial and non-spatial effect for a given cell s and β is the coefficients for all the effects. Specifically, the x_{ij} would contain the nine spatial neighborhood effect in velocity, airspeed, heading, vertical heading, peak mass, mass, mass correction, coordinates and position count, and those non-spatial features such as: visibility, cloud layer for coverage, average forecasted winddirection, average forecasted windspeed, average forecasted windspeed including gust and forecasted showers of rain, snow, thunderstorms, hours of the time , time slot of the day and the cell location indicator.

It is a natural choice for me to start from penalized regression model since there are too many features here and some of them are strongly correlated. The shrinkage methods I implemented include:

Ridge regression: $\hat{\beta}^{ridge} = \underset{\beta}{\operatorname{argmin}}\{\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2\}$

Lasso regression: $\hat{\beta}^{lasso} = \underset{\beta}{\operatorname{argmin}}\{\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|\}$

Elastic regression: $\hat{\beta}^{elastic} = \underset{\beta}{\operatorname{argmin}}\{\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2\}$

Adaptive Lasso regression: $\hat{\beta}^{adlasso} = \underset{\beta}{\operatorname{argmin}}\{\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \frac{1}{|\hat{\beta}_j|^\gamma}\}$
where β_j is the ordinary least squares estimate and $\gamma > 0$[3]

$$\text{MCP: } \hat{\beta}^{mcp} = \underset{\beta}{\operatorname{argmin}}\{\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \sum_{j=1}^{p} \lambda \int_0^{|\beta_j|} (1 - \frac{x}{(\gamma\lambda)})_+ \, \mathrm{d}x\} \text{ where } \gamma > 0[4]$$

$$\text{SCAD: } \hat{\beta}^{scad} = \underset{\beta}{\operatorname{argmin}}\{\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \begin{cases} \lambda \sum_{j=1}^{p} |\beta_j| & \text{if} |\beta_j| \leq \lambda \\ \sum_{j=1}^{p} -\frac{|\beta_j|^2 - 2a\lambda|\beta_j| + \lambda^2}{2(a-1)} & \text{if } \lambda < |\beta_j| \leq a\lambda \\ \sum_{j=1}^{p} \frac{(a+1)\lambda^2}{2} & \text{if } |\beta_j| > a\lambda \end{cases}\}$$

where $a > 2$ and $\lambda > 0$ [5]

Instead of cross validation, progress validation was performed to choose best penalized parameter because this is the time dependent data. The cross validation folds are randomly assigned, and it is not feasible that the training set contains information that occurs after the test set. So the procedure would be something like this:

- fold 1 : training (1), test (2)
- fold 2 : training (1 2), test (3)
- fold 3 : training (1 2 3), test (4)
- fold 4 : training (1 2 3 4), test (5)
- fold 5 : training (1 2 3 4 5), test (6)

...

The R and Python both provide parallel operation package to boost the running time. Neural Network was performed in the next step to check if the performance could be improved. The output layer could either be k class classification or regression value.From input layer to hidden layer, we could use an activation function to get the combinations of the units. For k class classification,there are k units in the output layers, and typically, when k=1, it shifts to the regression. Suppose a three-layers example, derived features Z_m are created from combinations of the inputs,and then the target Y_k (mean mass here) is modeled as a function of combinations of the Z_m,

$Z_m = \sigma(\alpha_{0m} + {\alpha_m}^T X), m = 1, \dots, M$ where X is all the spatial and non-spatial effects in the model

$T_k = \beta_{0k} + {\beta_k}^T Z, k = 1, \dots, K$

$Y_k = g_k(T), k = 1, \dots, K$ where $Z = (Z_1, Z_2, \dots, Z_M)$ and $T = (T_1, T_2, \dots, T_k)$[3]

The activation function is chosen to be the Tanh for classification and Rectifier for regression in H2O package.

$f(z) = tanh(z) = \frac{\exp(z) - \exp(z)}{\exp(z) + \exp(z)}$

$f(x) = max(0, x)$

H2O package[6] both in R and Python is the best choice, which could provide more arguments and parallel operation available. H2O, which is established by Java, is a multi-node cluster with shared memory model. The layers of neural networks were implemented from two to twelve in terms of this powerful package and the powerful cluster cloud platform. When predicting the mass value, we assume it follows the Gaussian distribution.Through setting cut-off value, I shift the regression problem into classification in order to show how likely the mass would be, which would follow Bernoulli distribution. The smaller the mass is, the less danger the birds would lead to. The bird control team did not care about the exact mass value of the birds while they only prefer to know where they should go. Because of such an enormous data set, the support vector machine proved useless. The logistic regression with regularization and deep neural network are implemented on the classification problem as well. All of the code for the algorithm and methods was written in R and Python, as well as the data manipulation. Some of the primary data acquisition and processing was written in Spark. During the whole project, I kept contact with the coordinator of bird control team to discuss the progress of my project routinely. Two demos were presented during the internship in front of the whole department so as to get more inspiration and suggestion from brain storm.

2.4 Results

The three-layer neural network model shows better than those penalized regression model. Nevertheless, it is still not the best choice since the average value and medium value of mean mass are approximately 11 and the root mean square error and mean absolute error are still 6.4 and 4.6 respectively. It didn't exhibit a competitive prediction ability. Some of the results shows below:

Table 2.1: Model Results on predicting Mean Mass

Methods	Test RMSE	Test MAE
RIDGE	6.521	4.811
ELASTIC	6.519	4.802
LASSO	6.519	4.801
Neural Networks (20-84-62)	6.429	4.600
Neural Networks (127-180-10)	6.409	4.649

Then we found the mass value prediction would easily underestimated the extreme value and the bird control team also indicated that they were actually not interested in the exact mass value the bird are. Thus, I prefer to set a cut-off value to shift the regression question to classification question and to show how likely the mass would be in a certain range. The four-layer neural network stands out among two to twelve layers neural network and it gives more than 75% accuracy rate in the test set. The best four-layer neural network shows below:

Table 2.2: 4-layer Neural Network Performance

Nodes (epochs=1)	Test Accuracy rate
195-188-4-100	0.752
128-71-26-65	0.753
106-5-183-41	0.753
83-35-150-118	0.754
83-12-63-156	0.755
195-39-121-193	0.751

The results in detail could be found in Appendix B Presentation.

2.5 Others

Moreover, I also prepared the data for Kaggle and Hackathon competition. The data is designed not only to meet the requirements of the competition, also to stimulate the creativity and imagination of those competitors, which may allow us gain a variety of ideas. The data structure for the competition essentially is quite different from mine since we should make sure those competitor are not able to get results directly from comparing the raw data set and the aggregated data set.

Chapter 3

Skills

3.1 Infrastructure

The whole project is implemented on the Cloudera Impala running on Apache Hadoop with data science necessary tools such as Jupyter notebook, R and Python. The fundamental knowledge I would learn is the basic operation on Impala query and Jupyter notebook. I grasp a basic understanding on the big data infrastructure. Hadoop is essentially an open source framework for processing, storing and analyzing data. Cloud computing would increase efficiency and help to collaborate in a team. I also configure the docker which is the worlds leading software containerization platform in my own computer to run python on it. With respect to data manipulation, I need to learn pyspark to do primary data acquisition and processing. The knowledge is totally fresh for me. Its very interesting to know this latest stuff and understand the structure and environment of big data project in business area. It could also allow me to know how to start my own data science project in my spare time.

3.2 Tools

In the perspective of data aggregation per location and per time step in grid, this work was both implemented in R and python. Since I am familiar with R and totally new on python, all the code was completed in R first and then transferred to python equivalently which help me not only to enhance the R skills, also to master another popular data science tools. The penalized regression I implemented in R include but not limited to those we learned from statistical learning course, for instance ridge regression, lasso regression, elastic regression[7]. It also contains adaptive lasso, minimax concave penalty[4], smoothly clipped absolute deviation penalty[5]. Those grouped regularization methods were also applied into searching the best results, like group lasso, group ridge etc. Thus, I must be acquainted with all of those algorithms and their implementations in R and Python. Afterwards, the deep neural network was introduced into it. The four layers with random nodes stand out among two to twelve layers test. The results from regression underestimated those extreme values which may result into some problem on the decision. For example, those underestimated large birds may threaten the aircraft more than those normal birds which are correctly estimated. After discussion with the bird control team, they actually dont need to know the exact value the bird mass is and those big birds are always the priority. Consequently, I dichotomize continuous mass variable and set a cut off value, shifting the regression question to classification question and implement the penalized logistic regression, support vector machine and neural networks to look for the best performance. I strengthened the use of those methods and study the implementation of H2O package mainly. The plenty of online tutorial could allow me to develop and and refine my understanding of application on this package and deep neural networks. Having said that, the included algorithms in H2O (Deep Neural Networks, Random Forest, GBM, K-Means, PCA etc) are solid for most of the common data mining tasks. With the superb memory management and the full integration with multi-node big data platforms, I am sure H2O engine is the most powerful deep learning tools. To speed up the computation, the parallel operation knowledge was also covered here. Some of the R packages did not provide the parallel computing, so the packages providing parallel facilities make it remarkably easy like doParallel. It must be a crucial knowledge for dealing with extremely large data set.

3.3 Others

The Data Innovation Lab team has experienced data scientists who are proficient in python, spark and those data infrastructures. Their inspiration, suggestion and ideas give me a solid understanding on how to deal with a business data project. In the meantime, they help me to cross the threshold on Python. It is very important to make full use of the resource around me and upgrade myself in a powerful team. The two demos I presented to the whole department help me to improve the effective communication skills as well. In order to present an easy understanding and non-academic demos, I must translate machine learning terms into words that the ordinary people can understand and provide some concrete examples which could show their relevance to the projects. I also join in many other project demos from other teams to broaden my horizon. My team leader arranged a bird control tour to experience how those guys work and close observe the moving behavior of birds. It helps me to thoroughly understand the data we have and the need they require.

Chapter 4

Conclusion

It is an extremely great experience for a foreign student to do an internship in *Schiphol Group*. I am really appreciated Data innovation lab in *Schiphol Group* could give me this opportunity of a life-time to apply theory into practice. Data is the foundation of any business, especially for the innovation and artificial intelligence area. Many unexpected challenge and obstacle would be met in the practical application fields. In this circumstance, knowledge and experience are particularly important. The knowledge from school could develop analytical and problem solving skills. Nevertheless, how to link the knowledge with the practical closely is only to be fostered in practice. Data Innovation Lab gives a latest data science platform while I could be instructed and mentored by experienced data scientist as well. Data science is keeping rapidly development, so too many new skills and packages ought to be acquired and mastered so as to keep up with the pace of the times. New knowledge and skills could be developed and strengthened constantly during the different kind of data science projects in business fields. it must be admitted that the gain in this half year internship grows quadratically. Referring to this project, I think the model based on Markov Chain may be another choice to improve the performance. XGboost[8], which is widely used for kaggle competitions and gives good results for most data sets, may be an alternative algorithm. Furthermore, there should be some other ideas to solve this problem and kaggle competition would inspire and surprise me. If the model would be implemented into user interface, how to make a real time project and boost the computing speed is still the challenge. In my experience, it would take such a long time to search the best performance results from one month data (468 features, 357080 Observations) even if we have a quite power cluster machine which means we could not updated the parameters of model in real time. However the bird control team needs the timely prediction data as reference to take measures immediately. Indeed, it is an interesting project and great experience, leading me to the data science fields. According to Harvard Business review, data scientist is the sexiest job of the 21st century[9]. The road to a good data scientist just started.

Bibliography

[1] Schiphol.nl.(2016) *online Available at: http://www.schiphol.nl/SchipholGroup/Company1/Profile.htm* [Accessed 5 Aug. 2016].

[2] Waldo R. Tobler *A computer movie simulating urban growth in the Detroit region* 1970.

[3] Jerome H. Friedman, Trevor Hastie, Rob Tibshirani *The Elements of Statistical Learning,Data Mining,Inference and Prediction* 2009.

[4] Cun-Hui Zhang *Nearly Unbiased Variable Selection Under Minimax Concave Penalty* 2010.

[5] J Fan *Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties* 2001.

[6] H2O.AI - AI FOR BUSINESS *online Available at: http://www.h2o.ai/docs/* [Accessed 5 Aug. 2016].

[7] Jerome H. Friedman, Trevor Hastie, Rob Tibshirani *Regularization Paths for Generalized Linear Models via Coordinate Descent* 2010.

[8] T Chen *XGBoost: A Scalable Tree Boosting System* 2016.

[9] Harvard Business Review. (2012) *online Available at: https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century* [Accessed 5 Aug. 2016].

Appendix A

Python Code

Birds Project

```
#import package
import findspark
findspark.init('/data/spark-1.6.0-bin-hadoop2.6')
from pyspark import SparkContext, HiveContext
from pyspark.sql import functions as F
from pyspark.sql import Window as w
%matplotlib inline
import seaborn as sns
import datetime as datetime
import pandas as pd
import numpy as np
import math
import time
from pyspark.sql.functions import UserDefinedFunction
from pyspark.sql.types import StringType, DoubleType, IntegerType
from shapely.wkb import loads
from shapely import wkt
from numpy.lib.stride_tricks import as_strided
from pyspark.sql import SQLContext
```

```
# load Spark and HiveContext
sc = SparkContext()
hc = HiveContext(sc)
```

Set the area of Polderbaan and calculate the max relative distance of target bird in each trajectory
Extend the Polderbaan area in terms of the max distance

```
# the area of Polderbaan
l_lon = 4.706
t_lat = 52.368
r_lon = 4.717
b_lat = 52.325
dlon =  r_lon-l_lon
dlat =  t_lat-b_lat
length0 =  r_lon*111.699 * np.cos(t_lat * np.pi/180) - l_lon*111.699 * np.cos(t_lat
* np.pi/180)
length = (int(np.ceil(length0*10)))/10.
width0 =  t_lat*110.574 - b_lat*110.574
width =  (int(np.ceil(width0*10)))/10.

# the potential extended distance in terms of the max relative distance in each
trajectory
# read track table which contanis trajectory and only extract the high risk birds
track_table = "birds.track"
track = (hc.read.table(track_table).where("classification_id != 1")
                                   .where("classification_id != 4")
                                   .where("classification_id != 5")
                                   .where("classification_id != 9")
                                   .where("classification_id != 10"))

# define the function for calculate the max relative distance on the basis of a
```

```
spherical earth
def max_dist(b):
    return max(((np.array(b)[1:,0]*111.321*np.cos(np.array(b)[1:,1]*np.pi/180)
                 - b[0][0]*111.321*np.cos(b[0][1]*np.pi/180))**2 +
                ((np.array(b)[1:,1] - b[0][1])*111)**2)**.5)

def do_something_to_cell(geo_string):
    return np.array([cell.split(' ') for cell in
str(geo_string[14:-1]).split(',')]).astype(float)

udf = UserDefinedFunction(lambda x: str(max_dist(do_something_to_cell(x))),
StringType())


track_trajectory = (track

.select('st_astext','classification_id','timestamp_start','timestamp_end')
                       .dropna()
                       .withColumn('max_dist', udf(F.col('st_astext')))
                       .withColumn('max_dist', F.col('max_dist').astype('float'))
                       .select('max_dist','classification_id')
                   )

nth_percentile = math.ceil(track_trajectory
                    .sort(track_trajectory.max_dist.desc())
                    .limit(int((1-0.95) * track_trajectory.count()))
                    .sort(track_trajectory.max_dist.asc())
                    .first()[0]
                          )
# calculate the 95% largest distance and to extend the exist area in terms of this
value
# drop those crazy large distance

# extend the area in terms of the max relative distance
lat_det = nth_percentile/111
lon_det = nth_percentile/(np.cos(b_lat*np.pi/180)*111.321)
bound_x = [(l_lon-lon_det,b_lat-lat_det)]
bound_y = [(r_lon+lon_det,t_lat+lat_det)]
print nth_percentile
```

Divided the extended area into grid (6 by 4)

```
x_n= 4
y_n= 6

x_cells_index = pd.RangeIndex(0,x_n,1)
y_cells_index = pd.RangeIndex(0,y_n,1)
x_cells=pd.qcut(([l_lon-lon_det,r_lon+lon_det]),x_n,retbins=True)[1:]
y_cells=pd.qcut(([b_lat-lat_det,t_lat+lat_det]),y_n,retbins=True)[1:]
# determine the bound of each cells and the index

bins_x=np.array(x_cells).tolist()[0]
bins_y=np.array(y_cells).tolist()[0]
# get the bound value
interval_lon = (r_lon+lon_det-(l_lon-lon_det))/x_n
```

```
interval_lat = (t_lat+lat_det-(b_lat-lat_det))/y_n
min_lon = l_lon-lon_det
min_lat = b_lat-lat_det
```

Get the coordinates of each position

```
# read trackestimate table which contanis each location
trackestimate_table = "birds.trackestimate"
trackestimate = hc.read.table(trackestimate_table)
track_subset =trackestimate.persist()
# transform the time format to drop those half seconds
trackestimate_subset = track_subset.withColumn('dt', F.date_format('timestamp',
'yyyy-MM-dd HH:mm'))
# define function to get the coordinates
# udf_x = UserDefinedFunction(lambda x:
str(loads(x,hex=True).__geo_interface__['coordinates'][0]), StringType())
# udf_y = UserDefinedFunction(lambda x:
str(loads(x,hex=True).__geo_interface__['coordinates'][1]), StringType())
def do_something_to_cell(geo_string):
    return [cell.split(' ') for cell in str(geo_string[9:-1]).split(' ')]

udf_x = UserDefinedFunction(lambda x: do_something_to_cell(x)[0][0], StringType())
udf_y = UserDefinedFunction(lambda x: do_something_to_cell(x)[1][0], StringType())
# transform the coordinates and the datatype
trackestimate_subset_coord=trackestimate_subset.withColumn('position_x',
udf_x(F.col('st_astext')).astype('float'))
trackestimate_subset_coord=trackestimate_subset_coord.withColumn('position_y',udf_y
(F.col('st_astext')).astype('float'))
```

Assign the coordinates into cells

```
# read track table which contanis trajectory and join the classification
information with each coordinates
track_table = "birds.track"
track = hc.read.table(track_table) # input track table which contanis trajectory
track = track.select('id','classification_id')
track_join = (trackestimate_subset_coord
            .join(track,on=track.id==trackestimate_subset_coord.track_id)
            .drop(track.id)
            .drop(trackestimate_subset_coord.st_astext)
            )
track_grid = (track_join.filter(track_join.position_x > bound_x[0][0])
                .filter(track_join.position_x < bound_y[0][0])
                .filter(track_join.position_y > bound_x[0][1])
                .filter(track_join.position_y < bound_y[0][1])).persist()
track_grid = track_grid[track_grid['classification_id'].isin(2,3,6,7,8)]
```

```

```

```
# check the mean position change to see if the change is stable or not
data_kernal = track_grid.select("dt","position_x","position_y")
data_kernal = data_kernal.withColumn('date', F.date_format('dt', 'yyyy-MM-dd'))
data_kernal_mean=data_kernal.groupBy('date').mean('position_x','position_y')
data_kernal_group=(data_kernal.join(data_kernal_mean,data_kernal.date==data_kernal_
mean.date,"inner").drop(data_kernal_mean.date)

.withColumn('subX',(F.col('position_x')-F.col('avg(position_x)'))**2)

.withColumn('subY',(F.col('position_y')-F.col('avg(position_y)'))**2)
                   )
data_kernal_group=data_kernal_group.groupBy('date').mean('subX','subY')
data_kernal_group=data_kernal_group.withColumn('sdist',(F.col('avg(subX)')+F.col('a
vg(subY)'))**0.5)
kernal =
data_kernal_group.join(data_kernal_mean,data_kernal_group.date==data_kernal_mean.da
te).drop(data_kernal_mean.date)
kernal = kernal.withColumn('dates', F.date_format('date', 'yyyy-MM-dd'))
```

Aggregate time into one minutes and join all the features

```
# assign each location to the cells index and aggregate to each minutes
track_grid_x = track_grid.withColumn('x_categories', F.ceil((F.col('position_x') -
min_lon)/interval_lon))
data = track_grid_x.withColumn('y_categories', F.ceil((F.col('position_y') -
min_lat)/interval_lat))
data = data.fillna(0).drop('radar_id')
data =
data.withColumn("location_index",F.concat(data.y_categories,data.x_categories)).dro
p('x_categories').drop('y_categories')
# aggregate to each minutes and join the attribute features
data_count=data.groupBy('location_index', 'dt').count()
attribute=data.groupBy('location_index', 'dt').mean('position_x','position_y',
                                                     'velocity','airspeed',
                                                     'heading','heading_vertical',

'peak_mass','mass','mass_correction')
cond = [data_count.location_index == attribute.location_index,
        data_count.dt == attribute.dt]
data_grid = (attribute.join(data_count, cond, 'inner')
                 .drop(attribute.dt)
                 .drop(attribute.location_index)
             )
data_grid = data_grid.orderBy('dt','location_index')
oldColumns = data_grid.schema.names
newColumns =
["position_x","position_y","velocity","airspeed","heading","heading_vertical",
              "peak_mass","mass","mass_correction","location_index","dt","count"]

data_grid_new = reduce(lambda data_grid, idx:
data_grid.withColumnRenamed(oldColumns[idx], newColumns[idx]),
xrange(len(oldColumns)), data_grid)
```

```
data_grid_new = data_grid_new.where("dt < '2016-04-20 00:00:00'")
```

EXAMPLE

Reshape and cast count as example and aggregate it into neighboorhood cell information structure

```
# count example
# extract the count of location points
grid_space = data_grid_new.select('location_index','dt','count').withColumn('dt',
F.date_format('dt', 'yyyy-MM-dd HH:mm:ss'))
grid_space =
grid_space.groupBy('dt').pivot('location_index').avg('count').fillna(0).sort('dt')



data = grid_space.toPandas()
def transform(arr,time):
    submatrix = []
    for i in range(0,4):
        for j in range(0,2):
            submatrix.append((arr.reshape(6,4)[i:i+3,j:j+3]).flatten())
    submatrix = np.array(submatrix)
    location = ("loc22","loc23","loc32","loc33","loc42","loc43","loc52","loc53")
    timestep = np.repeat(time,8)
    location = np.array(location)
    neighboor = np.c_[timestep,location,submatrix]
    return neighboor
nrow = data.shape[0]
trans=[]
for k in range(0,nrow):
        trans.append(transform(data.iloc[k,1:],data.iloc[k,0]))
df = np.concatenate(np.array(trans))
dfindex = pd.DataFrame(df).iloc[:,0:2]
zeros = np.zeros((8,9))
df1=pd.DataFrame(np.concatenate((zeros,df[:-8,2:])))
df2=pd.DataFrame(np.concatenate((zeros,zeros,df[:-16,2:])))
df3=pd.DataFrame(np.concatenate((zeros,zeros,zeros,df[:-24,2:])))
df4=pd.DataFrame(np.concatenate((zeros,zeros,zeros,zeros,df[:-32,2:])))
df5=pd.DataFrame(np.concatenate((zeros,zeros,zeros,zeros,zeros,df[:-40,2:])))
alldf = pd.concat([dfindex,df1, df2, df3, df4, df5],
axis=1).iloc[40:,].reset_index(drop=True)
alldf.columns = ['timestep', 'location',
                 'lt1','lm1','lb1','mt1','mm1','mb1','rt1','rm1','rb1',
                 'lt2','lm2','lb2','mt2','mm2','mb2','rt2','rm2','rb2',
                 'lt3','lm3','lb3','mt3','mm3','mb3','rt3','rm3','rb3',
                 'lt4','lm4','lb4','mt4','mm4','mb4','rt4','rm4','rb4',
                 'lt5','lm5','lb5','mt5','mm5','mb5','rt5','rm5','rb5']
```

Aggregate all features into neighboorhood cell information structure

```
attr = data_grid_new.toPandas()
attr=attr.rename(columns = {'dt':'timestep'})
```

```
attr['timestep'] =  pd.to_datetime(attr['timestep'], format='%Y-%m-%d %H:%M:%S')
attr['location_index'] = "loc"+ attr['location_index'].astype(str)
time =pd.date_range(attr['timestep'].min(),attr['timestep'].max(), freq='min')
timestep = pd.DataFrame(np.repeat(time, 24))
timestep = pd.DataFrame(np.repeat(time, 24))
loc = ("loc11","loc12","loc13","loc14",
       "loc21","loc22","loc23","loc24",
       "loc31","loc32","loc33","loc34",
       "loc41","loc42","loc43","loc44",
       "loc51","loc52","loc53","loc54",
       "loc61","loc62","loc63","loc64")
location=pd.DataFrame(np.tile(loc,time.shape[0]))
datapre = pd.concat((timestep,location),axis=1)
datapre.columns=['timestep','location_index']
attr['location_index'] = attr['location_index'].astype(str)
alldata = pd.merge(datapre, attr,
on=['location_index','timestep'],how="left").fillna(0)
```

```
ts =pd.date_range(attr['timestep'].min() +
pd.Timedelta(minutes=5),attr['timestep'].max(), freq='min')
timestep = pd.DataFrame(np.repeat(ts, 8))
loc = ("loc22","loc23",
       "loc32","loc33",
       "loc42","loc43",
       "loc52","loc53")
timediff = attr['timestep'].max() - (attr['timestep'].min() +
pd.Timedelta(minutes=5))
timediff = timediff / np.timedelta64(1, 'm') +1
location=pd.DataFrame(np.tile(loc,timediff))
pre = pd.concat((timestep,location),axis=1)
pre.columns=['timestep','location_index']
```

```
def transform(arr,time):
    submatrix = []
    for i in range(0,4):
        for j in range(0,2):
            submatrix.append((arr.reshape(6,4)[i:i+3,j:j+3]).flatten())
    submatrix = np.array(submatrix)
    location = ("loc22","loc23","loc32","loc33","loc42","loc43","loc52","loc53")
    timestep = np.repeat(time,8)
    location = np.array(location)
    neighboor = np.c_[timestep,location,submatrix]
    return neighboor


for m in range(2,alldata.shape[1]):
    col = [0,1,m]
    subdata = alldata.iloc[:,col]
    subdata = subdata.pivot(index='timestep', columns='location_index',
values=subdata.columns[2])
    subdata.reset_index(inplace=True)
    nrow = subdata.shape[0]
    trans=[]
    for k in range(0,nrow):
        trans.append(transform(subdata.iloc[k,1:],subdata.iloc[k,0]))
    df = np.concatenate(np.array(trans))
    dfindex = pd.DataFrame(df).iloc[:,0:2]
    zeros = np.zeros((8,9))
    df1=pd.DataFrame(np.concatenate((zeros,df[:-8,2:])))
    df2=pd.DataFrame(np.concatenate((zeros,zeros,df[:-16,2:])))
    df3=pd.DataFrame(np.concatenate((zeros,zeros,zeros,df[:-24,2:])))
    df4=pd.DataFrame(np.concatenate((zeros,zeros,zeros,zeros,df[:-32,2:])))
    df5=pd.DataFrame(np.concatenate((zeros,zeros,zeros,zeros,zeros,df[:-40,2:])))
    alldf = pd.concat([dfindex,df1, df2, df3, df4, df5], axis=1).iloc[40:,2:]
    columns =['lt1','lm1','lb1','mt1','mm1','mb1','rt1','rm1','rb1',
              'lt2','lm2','lb2','mt2','mm2','mb2','rt2','rm2','rb2',
              'lt3','lm3','lb3','mt3','mm3','mb3','rt3','rm3','rb3',
              'lt4','lm4','lb4','mt4','mm4','mb4','rt4','rm4','rb4',
              'lt5','lm5','lb5','mt5','mm5','mb5','rt5','rm5','rb5']
    columns = [s + alldata.columns.values[m] for s in columns]
    alldf.columns = columns
    pre = pd.concat([pre,alldf.reset_index(drop=True)],axis=1)
```

```
total =
pre.merge(attr[['location_index','timestep','peak_mass']],how="left",on=['location_
index','timestep']).fillna(0)
```

Join weather information to the features

```
# weather table
ts =pd.date_range(attr['timestep'].min() +
pd.Timedelta(minutes=5),attr['timestep'].max(), freq='min')
timediff = attr['timestep'].max() - (attr['timestep'].min() +
pd.Timedelta(minutes=5))
timediff = timediff / np.timedelta64(1, 'm') +1
index =range(timediff.astype('int'))
timestep = pd.DataFrame({'timestep': np.repeat(ts, 8),'index': np.repeat(index, 8)})
loc = ("loc22","loc23",
       "loc32","loc33",
       "loc42","loc43",
       "loc52","loc53")
location=pd.DataFrame(np.tile(loc,timediff))
pre_weather = pd.concat((timestep,location),axis=1)
pre_weather.columns=['index','timestep','location_index']
pre_weather['hour'] =pre_weather.timestep.apply(lambda x:pd.to_datetime(x).hour)
pre_weather['day_index'] = pre_weather.hour.apply(lambda x:
math.trunc(x/6)).astype('category')
pre_weather['tenmin'] = pre_weather.timestep.apply(lambda x:
                                          x -
datetime.timedelta(minutes=pd.to_datetime(x).minute
                                                                   -
math.trunc(pd.to_datetime(x).minute/10)*10))

weather=pd.read_csv('/data/capdec/raw/capdec/weather_2016.txt',sep='\t')
weather = weather[["DATE-LT","TIME-LT","VIS","CEIL","GML","BZO","
~VIS","~CEIL","~GML","~WDIR","~WSPD","~WGUS","~SHWR"]]
```

```
weather['DATE-LT'] = pd.to_datetime(weather['DATE-LT'],format='%d-%m-%Y')
weather['tenmin'] =  pd.to_datetime(pd.to_datetime(weather['DATE-LT'].astype(str) +
' ' + weather['TIME-LT']))
weather = weather.drop('TIME-LT',axis=1).drop('DATE-LT',axis=1)
weather_join = pd.merge(pre_weather, weather,on='tenmin',how='left')
```

```
birrrds = pd.merge(total,weather_join,on=['location_index','timestep'],how='left')
```

```
# analysis h2o
## penalized regression
```

```
birrrds['folds'] = pd.cut(birrrds.index,20,labels=range(20))
```

```
import h2o
h2o.init()
```

Penalized Regression

```
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
from h2o.grid.grid_search import H2OGridSearch
features = birrrds.columns.drop(['timestep','folds','peak_mass']).tolist()
rmse = performance = all_models = best_lambda = best_performance = []
alpha_opts =  [0,0.25,0.5,0.75,1]
lam = np.arange(0, 1, 0.01)
for j, alpha in enumerate(alpha_opts):
    hyper_parameters =  alpha_opts[j]
    for k in range(0,100,1):
        Lambda = lam[k]
        for i in range(9):
            train = birrrds.loc[birrrds['folds'].isin(range(10+i))]
            valid =
birrrds.loc[birrrds['folds'].isin([i+11])].reset_index(drop=True)
            training =  h2o.H2OFrame((train.values.tolist()))
            validation = h2o.H2OFrame((valid.values.tolist()))
            training.names = train.columns.tolist()
            validation.names = valid.columns.tolist()
            models = H2OGeneralizedLinearEstimator(family = "gaussian",Lambda =
Lambda,standardize=True,
                                                alpha = hyper_parameters,
max_iterations= 100)
            models.train(y = 'peak_mass', x = features, training_frame =
training,validation_frame=validation)
            rmse = rmse + [np.sqrt(models.mse(valid=True))]
    performance = performance + [np.mean(rmse)]
    loc = np.argmin(performance) + 1
    b_lambda = lam[loc]
best_lambda =  best_lambda +  [b_lambda]
best_performance = best_performance + [min(performance)]
```

Neural networks in regression

```
# find best neural nets
rmse_NN = performance_NN = []
features = birrrds.columns.drop(['timestep','folds','peak_mass']).tolist()
from h2o.estimators.deeplearning import H2OAutoEncoderEstimator,
H2ODeepLearningEstimator
import random
for j in range(1):
    hidden = random.sample(range(1, 200), 4)
    for i in range(9):
        train = birrrds.loc[birrrds['folds'].isin(range(10+i))]
        valid = birrrds.loc[birrrds['folds'].isin([i+11])].reset_index(drop=True)
        training =  h2o.H2OFrame((train.values.tolist()))
        validation = h2o.H2OFrame((valid.values.tolist()))
        training.names = train.columns.tolist()
        validation.names = valid.columns.tolist()
        models_NN =
H2ODeepLearningEstimator(activation="Rectifier",hidden=hidden,distribution
="gaussian",epochs =10,
                                      loss ="Quadratic",l1=1e-05, max_w2=10 )
        models_NN.train(y = 'peak_mass', x = features, training_frame =
training,validation_frame=validation)
        rmse_NN = rmse_NN + [np.sqrt(models_NN.mse(valid=True))]
    performance_NN = performance_NN + [np.mean(rmse_NN)]
```

Classification in Logistic Regression & Neural Networks

```
# classification set the cut off value
cutoff = np.mean(birrrds.peak_mass)
maxcutoff = np.max(birrrds.peak_mass)
birrrds['binaryclass'] =
pd.cut(birrrds['peak_mass'],bins=[0,cutoff,maxcutoff],labels=['low','high'],include
_lowest=True)
birrrds_lr= birrrds.drop('tenmin', 1).drop('peak_mass', 1)
birrrds_lr =birrrds_lr.assign(binaryclass = lambda x:
pd.factorize(x.binaryclass)[0])
birrrds_lr =birrrds_lr.assign(location_index = lambda x:
pd.factorize(x.location_index)[0])
```

Logistic regression(

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import accuracy_score
```

```
accuracy = accurate = []
#c = np.arange(0.05, 1.05, 0.02)
#length = len(c)
#for k in range(0,length,1):
#    C = c[k]
for i in range(8):
    train = birrrds_lr.loc[birrrds_lr['folds'].isin(range(10+i))]
    valid = birrrds_lr.loc[birrrds_lr['folds'].isin([i+11])].reset_index(drop=True)
    X_lr = train.iloc[:,1:-2]
    Y_lr = train.iloc[:,-1]
    lrmodel = LogisticRegression(C=0.001, penalty='l1',max_iter =100)
    lrmodel = lrmodel.fit(X_lr,Y_lr)
    X_valid = valid.iloc[:,1:-2]
    predicted = lrmodel.predict(X_valid)
    expected = valid.iloc[:,-1]
#        accuracy = accuracy + [accuracy_score(predicted, expected)]
    table = predicted  == expected
    accuracy = table.sum()/ (expected.shape[0] * 1.0)
    accurate = accurate + [np.mean(accuracy)]
```

```
accurate
```

Find the best penalized parameter

```
finalmodel =  LogisticRegression(C=bestc, penalty='l1', tol=0.01)
train = birrrds.loc[birrrds['folds'].isin(range(10+8))]
valid = birrrds.loc[birrrds['folds'].isin([8+11])].reset_index(drop=True)
X_final = train.iloc[:,1:-2]
Y_final = train.iloc[:,-1]
finalmodel = finalmodel.fit(X_final,Y_final)
X_valid = valid.iloc[:,1:-2]
predicted = finalmodel.predict_proba(X_valid)
```

NN classification in H2O

```
cutoff = np.mean(birrrds.peak_mass)
maxcutoff = np.max(birrrds.peak_mass)
birrrds['binaryclass'] =
pd.cut(birrrds['peak_mass'],bins=[0,cutoff,maxcutoff],labels=['low','high'],include
_lowest=True)
```

```
features =
birrrds.columns.drop(['timestep','folds','binaryclass','peak_mass']).tolist()
logloss_NN = performance_NN = performance_auc = auc_NN = layer = accurate =[]
from h2o.estimators.deeplearning import H2OAutoEncoderEstimator,
H2ODeepLearningEstimator
import random
for j in range(1):
    # increase range number to try more nodes
    hidden = random.sample(range(1, 200), 4)
    for i in range(8):
        train = birrrds.loc[birrrds['folds'].isin(range(10+i))]
        valid = birrrds.loc[birrrds['folds'].isin([i+11])].reset_index(drop=True)
        training =  h2o.H2OFrame((train.values.tolist()))
        validation = h2o.H2OFrame((valid.values.tolist()))
        training.names = train.columns.tolist()
        validation.names = valid.columns.tolist()
        models_NN = 
H2ODeepLearningEstimator(activation="Tanh",hidden=hidden,distribution
="bernoulli",epochs =10,
                                      loss ="CrossEntropy",l2=1e-05)
        models_NN.train(y = 'binaryclass', x = features, training_frame = 
training,validation_frame=validation)
        logloss_NN = logloss_NN + [models_NN.logloss(valid=True)]
        auc_NN = auc_NN + [models_NN.auc(valid=True)]
        pred = models_NN.predict(validation)
        table = pred['predict'] == validation['binaryclass']
        accuracy = table.sum()/validation.shape[0]
    performance_NN = performance_NN + [np.mean(logloss_NN)]
    performance_auc = performance_auc + [np.mean(auc_NN)]
    accurate = accurate + [np.mean(accuracy)]
    layer = layer + hidden
```

```
accurate
```

```
# h2o logistic regression
features = 
birrrds.columns.drop(['timestep','folds','binaryclass','peak_mass']).tolist()
logloss_NN = performance_NN = performance_auc = auc_NN = layer = accurate =[]
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
from h2o.grid.grid_search import H2OGridSearch
rmse = performance = all_models = best_lambda = best_performance = []
alpha_opts =  [1]
#lam = np.arange(0, 1, 0.01)
for j, alpha in enumerate(alpha_opts):
    hyper_parameters =  alpha_opts[j]
#    for k in range(0,100,1):
    Lambda = 0.001
    for i in range(8):
        train = birrrds.loc[birrrds['folds'].isin(range(10+i))]
        valid = birrrds.loc[birrrds['folds'].isin([i+11])].reset_index(drop=True)
        training =  h2o.H2OFrame((train.values.tolist()))
        validation = h2o.H2OFrame((valid.values.tolist()))
        training.names = train.columns.tolist()
        validation.names = valid.columns.tolist()
        models = H2OGeneralizedLinearEstimator(family = "binomial",Lambda = 
Lambda,standardize=True,
                                                   alpha = hyper_parameters, 
max_iterations= 100)
        models.train(y = 'binaryclass', x = features, training_frame = 
training,validation_frame=validation)
        pred = models.predict(validation)
        table = pred['predict'] == validation['binaryclass']
        accuracy = table.sum()/validation.shape[0]
        accurate = accurate + [np.mean(accuracy)]
```

```
accurate
```

```
train_all = birrrds.loc[birrrds['folds'].isin(range(19))]
test = birrrds.loc[birrrds['folds'].isin([19])].reset_index(drop=True)
training_all =  h2o.H2OFrame(train.values.tolist())
tests = h2o.H2OFrame(test.values.tolist())
training_all.names = train_all.columns.tolist()
tests.names = test.columns.tolist()
models_NN = H2ODeepLearningEstimator(activation="Tanh",hidden=hidden,distribution
="bernoulli",epochs =10,
                                      loss ="CrossEntropy",l2=1e-05)
models_NN.train(y = 'binaryclass', x = features, training_frame = training_all)
pred = models_NN.predict(tests)
```

```
data = pd.concat([pd.DataFrame(test['binaryclass']),h2o.as_list(pred)],axis=1)
accuracy = sum(data['binaryclass'] == data['predict'])/round(data.shape[0])
```

```
h2o.shutdown()
```

```
birrrds.to_csv('data/birrrds.csv')
from IPython.display import FileLink, FileLinks
FileLinks('data')
```

Appendix B

Presentation