Road Accident Prediction in Time and Space

1st Antoine Hébert
Department of Computer Science and Software Engineering
Concordia University
Montréal, Québec, Canada
an_heb@encs.concordia.ca

1st Timothée Guédon
Department of Computer Science and Software Engineering
Concordia University
Montréal, Québec, Canada
t_guedon@encs.concordia.ca

3rd Tristan Glatard
Department of Computer Science and Software Engineering
Concordia University
Montréal, Québec, Canada
tglatard@encs.concordia.ca

4th Brigitte Jaumard
Department of Computer Science and Software Engineering
Concordia University
Montréal, Québec, Canada
bjaumard@cse.concordia.ca

***Abstract*—Road accidents are an important issue of our modern societies, responsible for millions of deaths and injuries. In this paper we show how one can leverage open datasets from a City like Montreal, Canada, to create some accident prediction models, using state-of-the-art big data analytics methods in Python. Such models could then be used in the context of road accident prevention, but also to identify key factors that can lead to a road accident, and eventually, help to elaborate new policies. This study also explains how we dealt with the severe class imbalance issue of the accident prediction problem, a recurrent issue that touches many other scientific domains like medical diagnoses or fraud detection. In particular, we show how we implemented Balanced Random Forests, a variant of the Random Forests machine learning algorithm, into the Scala and Python APIs of Apache Spark big data framework. All source code can be found on Github.**

***Index Terms*—machine learning, road accidents prediction**

I. Introduction

A. Context

According to the World report on road traffic injury prevention, 2004, more than one million people die every year from car accidents and dozens of millions of people are being injured [1]. In this report, the World Health Organization also describe road traffic systems as "the most complex and the most dangerous system with which people have to deal every day". This issue is all the more important that the number of accidents is forecast to continue rising. In the last decade, Big Data Analytics has been emerging [2], allowing scientists to handle a very large amount of data, but also complex and heterogeneous data, in order to extract useful insights from it. In the context of accident prediction, Big Data Analytics could provide insights on the conditions leading to an increased risk of road accidents that could then be used to develop traffic-related policies and prevention operations. Our contributions include: (1) a demonstration of how simple and easily available datasets can be assembled to obtain meaningful features for road accident prediction, (2) a comparison of three different machine learning algorithm dealing with data imbalance in the context of road accident prediction, (3) the implementation of Balanced Random Forest introduced by Chen et al. in Apache Spark for efficient distributed training. Finally, all the source code used is publicly available on Github under the MIT license.

B. Related work

Accident prediction has been extensively studied in the last decade. The main techniques that have been used at the beginning of the research in road accidents prediction were methods using the Poisson distribution and the negative binomial regression [3]. Then, as machine learning methods become more popular, algorithms like decision trees have been used in several forms like classification and regression decision trees, frequent pattern trees and random forests [3]–[6], along with other machine learning algorithms like Support Vector Machines [7]. Recently, the outstanding progress of deep learning algorithms have resulted in their extensive use in every field, including in Geospatial machine learning (Convolutional Neural Network and Long Short-Term Memory neural network for example) [8], [9].

II. Methods

A. Choosing a Big Data framework: Apache Spark or Dask

According to Dask's official website, "Dask is a flexible library for parallel computing in Python" [10]. Apache Spark, on the other hand, describes itself as "a fast and general-purpose cluster computing system" [11]. Both of them are accepted tools with wide supporting community, even though Apache Spark is a bigger and older project. Although we almost exclusively used Apache Spark's Python API (called "pySpark") in this study, we started our experiment using Dask. Some advantages of Dask over Spark made us chose it for our analysis in the first place; For example, Dask "is developed in coordination with [...] projects like Numpy, Pandas and Scikit-Learn" [10], all widely used Python packages in machine-learning and data science. It is also admitted that Dask is lighter than Spark and designed to be more flexible in terms of applications and algorithms. However, we found

ourselves writing a lot more code using Dask than Spark for the same result. Spark map-reduce-based interface is very handy and allow easier processing of datasets. For all these reasons we found out that Spark fitted our needs best and we switched to it at an early time of the study.

B. Algorithms selected

For this study, we chose to focus on tree-based algorithms because they have proven their efficiency compared to classical statistical methods and allow for easier interpretability than deep learning algorithms. In particular, we implemented the Balanced Random Forests (BRF) algorithm in Apache Spark to deal with class imbalance which is a prominent issue in road accident prediction. To validate the performance of our version, we run an experiment with a simple synthetic dataset to compare its performances with the standard implementation of Random Forest (RF) in Spark with random under-sampling of the majority class. Balanced Random Forest performed much better. Balanced Random Forests in one of the two methods proposed by Chen, Liaw, and Breiman to deal with class imbalance when using Random Forest. Balanced Random Forests are like Random Forests, but with a difference during the bootstrapping phase, for each tree of the forest, a random under-sampling of the majority class is performed in order to obtain a balanced sample. Intuitively, Balance Random Forest is an adaptation of random undersampling of the majority class making use of the fact that Random Forests are an ensemble method. Random undersampling usually performs better than more advanced methods like SMOTE or NearMiss [12]. Chen, Liaw and Breiman [13] also proposed in the same paper, the Weighted Random Forest (WRF) which consists in giving more weight to the minority class at the building time of each tree and during the measure of the gain in impurity and during the class prediction of each terminal node. While neither method is clearly better than the other in terms of prediction power, we chose BRF because it is found to be more computationally efficient (because of the under-sampling) and easier to implement in our opinion. Interestingly, Wallace et al. [14] presents a theoretical analysis of the data imbalance problem and suggest to use methods like Balance Random Forest.

We decided to compare the performance of the Balanced Random Forests algorithm with the performances of one of the very efficient implementations of Gradient Boosted Trees: XGBoost which offers parameters to deal with data imbalance as well.

C. Datasets

We used three public datasets provided by the city of Montreal and the government of Canada: (1) Montreal Vehicle Collision: This dataset provided by the city of Montreal contains all the road collisions reported by the police in Montreal from 2012 to 2018. For each accident, the dataset contains the date and localization of the accident, information on the number of injuries and death, the number of vehicles involved, and information on the road condition. We used only the localization and the date of the accident since we do not have the other piece of information when no accident happened. Another dataset with all vehicle collision in Canada is available but without the localization of the accident, therefore we restrained our analysis to the city of Montreal. (2) National Road Network: This dataset provided by the government of Canada contains the geometry of all roads in Canada. For each road segment, a few meta-data are given. Sadly, for roads in Quebec the speed limit and the surface type are not provided. The data was available in various format, we chose to use the Keyhole Markup Language, which is easy to read because it is based on the Extensible Markup Language (XML). (3) Data Historical Climate Data: This dataset provided by the government of Canada contains hourly weather information measured at different weather stations across Canada. For each station and every hour, the dataset provides the temperature, the humidity, the wind direction and speed, the visibility, the atmospheric pressure and observations of atmospheric phenomenon such as snow, fog, rain, etc.

D. Positive and negative samples generation

Our machine learning problem can be stated as a binary classification problem, the positive class being the occurrence of an accident and the negative class being all the cases where an accident did not happen. We got a little bit less than 150 000 positive samples from the national collision dataset and we generated a random sub-sample of 5 million negative samples (over 10 billion possibilities). The generation of the negative samples basically consisted in a Cartesian product between the roads from the road network dataset and all the hours of all days that occurred between the 1st January 2012 and the 31 of December 2017, the dates range of the Montreal Vehicle Collision dataset. This huge cartesian product operation together with the joins with the data from the other datasets made our dataset generation pretty heavy. It resulted in an impractical amount of time and memory space requirements to generate the dataset. Our first implementation was querying the Historical Climate Data API in real-time with a cache mechanism. Our idea was to collect only the weather stations and hours necessary for our sample of negative examples, but it resulted in bad performances. We got a performance increase by first building a Spark data frame with all the Historical Climate Data for weather stations around Montreal and then joining the two datasets. We conducted a detailed analysis of our algorithm to improve its performances. We notably obtained a good performance increase by not persisting intermediary results of the road segment identification for accidents. As opposed to what we initially thought, recomputing these results was faster than writing and reading them in the cache. Finally, the identification of the road segment corresponding to accidents was very memory intensive, we modified this step to be executed by batch of one month. With these improvements and a few other tricks including partitioning the data frame at key points in our algorithm, we manage to reduce the processing time from about one minute to generate about 1000 positive

samples to less than 1 second. We also used a cluster from "Compute Canada" to take maximum advantage of Apache Spark distributed nature for the generation of examples and the hyper-parameter tuning of our models.

E. Feature Engineering

Once we had our pairs of date with hour and road segment for positive and negative examples, we created features from this information. We created four types of features, weather features, features from the road segment, features from the date .

For the weather features, we used data from the Historical Climate Dataset. In order to estimate the weather information at the position of the road segment we compute the mean of the weather information from all the surrounding weather stations at the date and hour of the example weighted by the inverse of the square of the distance between the station and the road segment. We initially used the inverse of the distance, but we obtained a small improvement in performances when squaring the inverse of the distance. It makes sense since, the weather information is probably much closer to the closest stations than the other ones. We tried higher power, but results were not as good. It could be interesting to experiment with different power values for each weather information. We used all the weather information provided by the Historical Climate Dataset. The features are: the temperature, the dew point temperature which is a measure of the humidity, the real humidty percentage, the wind direction, the wind speed, the visibility, the atmospheric pressure, the Hmdx index, which is a measure of felt temperature and the wind chill which is another measure of the felt temperature using the wind information. In addition, the historical weather dataset, provide a multi-label categorical variable providing the observations of atmospheric phenomenon such as snow, fog, rain, etc. We used this last variable to create a binary variable that we called risky weather which is true when the following phenomenon are observed: Freezing Rain, Freezing Drizzle, Snow, Snow Grains, Ice Crystals, Ice Pellets, Ice Pellet Showers, Snow Showers, Snow Pellets, Ice Fog, Blowing Snow, Freezing Fog.

For the features from the road segments, we were limited by the limited metadata provided on the road segments. From the shape of the road segment, we computed the length of the road segment, and from the name of the street we identified the type of road (Highway, street, boulevard, etc.). In addition road segment are classified in three different level in the dataset depending on their importance in the road network, we created a categorical feature from this information. For these two categorical features, we encoded them as suggested in Element of Statistical Learning [15] in section 9.2.4, instead of using one hot encoding which would create an exponential number of splits, we index the categorical variable ordered by the proportion of the example belonging to the given category which are positive samples. This encoding guarantee to provide optimal splits on these categorical variables. Lastly, we added an a feature giving the number of accidents which occurred previously on this road segment.

For the date features, is is quite simple, we took the day of the year, the hour of the day, and the day of the week. We decided to encode day of the year and hour of the day to be cylic. Indeed, for example with hour of the day, 23 hour is very close to 1 hour, but in the usual encoding this does not appear to the machine learning algorithm. With cyclical encoding, we compute two features, the first one is the cosine of the original feature scaled between 0 and two pi, and the second one is the sine of the original feature scaled between 0 and two pi.

F. Implementation of balanced random forest in Apache Spark

The Balanced Random Forests algorithm was not implemented in Apache Spark. An implementation is available in the python library imbalanced-learn [16] which implements many algorithm to deal with data imbalance using an API inspired by scikit-learn, but the size of our dataset made it impossible for us to use this library. Therefore, we implemented Balance Random Forests in Apache Spark.

In the Apache Spark implementation of Random Forests, the bootstrap step is made before starting to grow any tree. For each sample, an array contains the number of time it will appear in each tree. When doing sampling with replacement, each value of this array is given by a Poisson distribution. The parameter of the Poisson distribution correspond to the subsampling rate hyper-parameter of the Random Forest, which specifies the size of the sample used for training each tree as a fraction of the total size of the dataset. Indeed, if for example we want each tree to use a sample of the same size of the whole dataset, the subsampling ratio will be set to 1.0, which is indeed the number of time a given example will appear in a tree on average.

In order to implement Balanced Random Forests, we modified the parameter of the Poisson distribution to use the class weight multiplied by the subsampling ratio, this way a negative sample with for example the weight 0.25 has 4 times less chance to be chosen to appear in a given tree. This implementation has the advantage that it did not required a big code change and will be easy to test, but it also has a disadvantage, users probably expect linearly correlated weight to be equivalent, which is not the case in our implementation.

In order to be compatible with other possible use cases, the weight are actually applied per samples and not per class. This is a choice made by Apache Spark developers that we respected. In order to support sample weights, we create a new Poisson distribution for each sample. To make sure the random number generator is not reseeded for each sample, we use the same underlying random number generator for all Poisson distributions, this also helps reduce the cost of creating a new Poisson distribution object. Like with other estimators accepting weights, our Balanced Random Forests implementation use weight from a weight column in the samples data frame. We adapted the Python wrapper of the random forest classifier to accept and forward weights to the algorithm in Scala. Finally, we did a first validation of our implementation using an artificially generated imbalanced

dataset and compared the results of both standard random forest and BRF. While BRF had better results in terms of area under PR (which was our selected measure) (98.% against 98.1%), standard random forest had better F1 score (99.7% against 98.8%) with default threshold. This emphasized that we were right in our choice of the area under PR curve as our measure in the case of class imbalance, indeed with a good threshold, the BRF model performs better.

G. Hyper-parameter tunings

In order to determine the optimal hyper parameter, we first performed automatic hyper parameter tuning using Apache Spark ML library grid search using cross validation. Because the training on the whole dataset would have been too high, we took a small sample of the dataset. Still, We could not test many parameters combination using this method.

Once we got a first tuning result with grid search we continued manually by following a plan, do , check, adjust method, using the measure of the area under the precision recall curve, the precision and recall with different threshold on the test set and the training set we were able to better understand how the performances of our model could be improved.

Interestingly, despite using many trees, our Random Forest algorithms tended to overfit very quickly as soon as the maximum depth parameter goes above 18. We eventually used only 100 trees, because more trees did not help much. We have not tried more than 200 trees, maybe many more trees would have been necessary to increase the maximum depth without over fitting, but then the training time would not be reasonable.

H. Choice of a metric to evaluate our models

The Chosen metric for the evaluation of the models was the area under Precision/Recall (PR) curve (area under PR). According to Davis and Goadrich [17] this measure is the most representative of how well a model has been trained in the case of data imbalance, particularly compared to the ROC curve which is the commonly used measure. One can intuitively be convinced because in the ROC curve the false positive rate (FPR) is used instead of the precision. The precision refers to the proportion of examples detected as positive which are actually positive, while the FPR refers to the proportion of negative detected as positives. With an imbalanced dataset, the FPR will usually stay very low because the number of negative is high. The precision, by contrast, will usually be quite low since algorithms tuned for data imbalance will usually include many false positives in the examples they detect in order to achieve a correct recall also called true positive rate.

I. Identifying the most important features

Random forests allow to measure feature importances by computing the total decrease in impurity of all splits using the variable weighted by the number of samples. This variable importance measure is not perfect for interpretability since it is biased toward non-correlated variables, but it helps to select the most useful features for the prediction. You can find the feature importances given by the Balance Random Forests in figure 1. According to this figure, the most important features to predict the occurrence of an accident is the number of accidents which occurred on this road during the previous years which we could have easily guessed. Then the location of the road, the type and length of the road segment and the hour of the day have a high importance. As compared to the count of accident feature the remaining features seem to have almost no importance, but the model does not perform as well if we remove them. Surprisingly, the day of the year, the temperature and the risky weather boolean have very low feature importance. We believe that the weather features have lower feature importances because they are strongly correlated. The gain in impurity the information they contain provide is distributed over all the correlated features. Also, the risky weather feature would probably require more feature engineering, since if it was snowing one hour ago, there is still a higher risk. We could do a moving average.

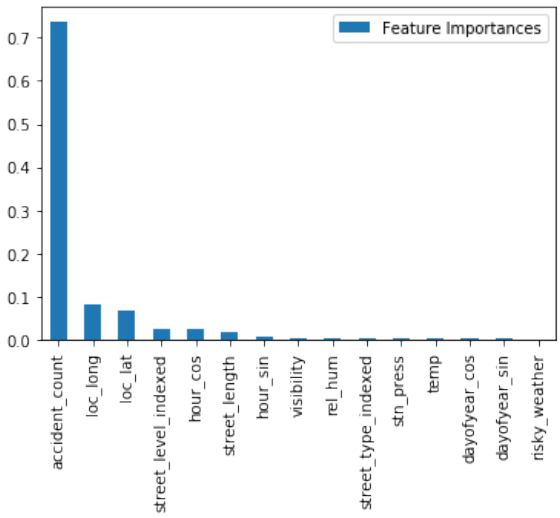

Fig. 1. Feature importances computed by the Balanced Random Forest

III. Results

The training time of the machine learning algorithm is very long due to the size of our dataset and the latest improvement in term of feature engineering have been discovered very late. Therefore, we currently only have the results with the Balanced Random Forest using the latest features. The results from the previous training of other algorithm are not comparable since they do not use the same features. We will add the missing results in later versions of this document.

These results have been obtained by training the Balanced Random Forest algorithm on the whole dataset of positive samples, but with only a fifth of the dataset of negative samples we have generated. Therefore the data imabalance is reduced to a factor of 8. According to our previous experiments, the results are almost the same when using the full dataset of negative samples, sometimes slightly better. To evaluate this model we used a test set containing the last two years of our

dataset, the model is trained on the 4 previous years and use only data from these years. For example, the "count_accident" feature contains only the count of accidents on road segments which occurred from 2012 to 2016.

We obtain an area under the precision-recall curve of 79%, and an area under the ROC curve of 97%. This results in the following precision and recall values depending on the chosen probability threshold:

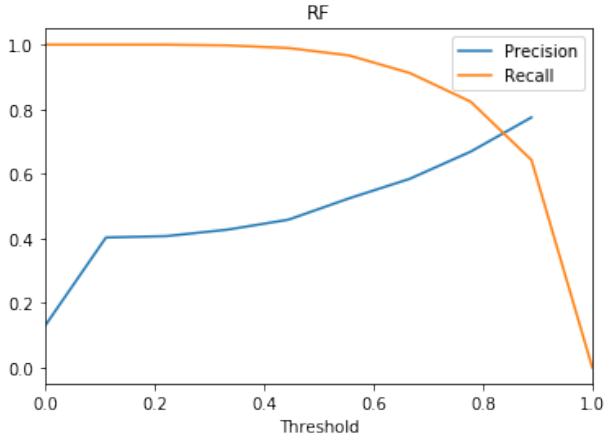

Fig. 2. Precision and recall as a function of the probablity threshold

As we can see, with a threshold set to 0.66, we can achieve a recall of 91% with a precision of 58%. With a threshold of 0.11, the model can actually detect almost all the accidents, with a precision of 40%.

Our source code is available on Github at https://github.com/GTimothee/accident-prediction-montreal.

IV. Discussion

The overall results are quite good but mostly rely on the count of previous accidents on the road segment. This is not an issue for accident prediction, but it does not help to understand why these roads are particularly dangerous. Accident prediction is a very hard machine learning problem because even if the risk is very high for an accident to occur, the driver might manage to avoid it in time and avoid the accident although the conditions were almost exactly the same as when an accident occurs. We would need information on the state of the drivers and the state of the vehicle to significantly improve performances.

However, we believe these results can be improved even without this important feature by adding more spatial features. We can see that one of the most important variables are the localization of the accident, but the ability to learn a spatial distribution by the random forest algorithm is limited. Therefore we believe providing more context information on where the road is located will greatly help. On another experiment we have seen the population density and the distance to the closest billboard used as features successfully. However, it will be hard to reach good performances because of the nature of the problem without the count of accidents.

A. Future work

We believe better performances could be reached by adding more features from other datasets. For the city of Montreal, we identified two particularly interesting datasets: a dataset with the location and dates of construction work on roads, and a dataset with the population density. These datasets might not be as easily available for over areas, but for Montreal, it could improve performances. Also, the solar elevation and the solar azimuth could easily be added without the need for another dataset. These features have been used successfully in road accident prediction in previous experiments.

Acknowledgment

We acknowledge Compute Canada for giving us access to the cluster.

References

[1] M. Peden, R. Scurfield, D. Sleet, D. Mohan, A. Hyder, E. Jarawan, and C. Mathers, "World report on road traffic injury prevention," *BMJ*, vol. 328, no. 7444, p. 846, 2004.

[2] A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *International Journal of Information Management*, vol. 35, no. 2, pp. 137 – 144, 2015.

[3] L.-Y. Chang and W.-C. Chen, "Data mining of tree-based models to analyze freeway accident frequency," *Journal of Safety Research*, vol. 36, no. 4, pp. 365 – 375, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022437505000708

[4] A. Theofilatos, "Incorporating real-time traffic and weather data to explore road accident likelihood and severity in urban arterials," *Journal of Safety Research*, vol. 61, pp. 9 – 21, 2017.

[5] J. Abellán, G. López, and J. de Oña, "Analysis of traffic accident severity using decision rules via decision trees," *Expert Systems with Applications*, vol. 40, no. 15, pp. 6047 – 6054, 2013.

[6] L. Lin, Q. Wang, and A. W. Sadek, "A novel variable selection method based on frequent pattern tree for real-time traffic accident risk prediction," *Transportation Research Part C: Emerging Technologies*, vol. 55, pp. 444 – 459, 2015, engineering and Applied Sciences Optimization (OPT-i) - Professor Matthew G. Karlaftis Memorial Issue.

[7] X. Li, D. Lord, Y. Zhang, and Y. Xie, "Predicting motor vehicle crashes using support vector machine models," *Accident Analysis & Prevention*, vol. 40, no. 4, pp. 1611 – 1618, 2008.

[8] Z. Yuan, X. Zhou, and T. Yang, "Hetero-convlstm: A deep learning approach to traffic accident prediction on heterogeneous spatio-temporal data," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '18. New York, NY, USA: ACM, 2018, pp. 984–992. [Online]. Available: http://doi.acm.org/10.1145/3219819.3219922

[9] Q. Chen, X. Song, H. Yamada, and R. Shibasaki, "Learning deep representation from big and heterogeneous data for traffic accident inference," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, pp. 338–344. [Online]. Available: http://dl.acm.org/citation.cfm?id=3015812.3015863

[10] Dask Development Team, *Dask: Library for dynamic task scheduling*, 2016. [Online]. Available: https://dask.org

[11] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016. [Online]. Available: http://doi.acm.org/10.1145/2934664

[12] P. Branco, L. Torgo, and R. P. Ribeiro, "A survey of predictive modeling on imbalanced domains," *ACM Comput. Surv.*, vol. 49, no. 2, pp. 31:1–31:50, Aug. 2016. [Online]. Available: http://doi.acm.org/10.1145/2907070

[13] C. Chen and L. Breiman, "Using random forest to learn imbalanced data," *University of California, Berkeley*, 01 2004.

[14] B. C. Wallace, K. Small, C. E. Brodley, and T. A. Trikalinos, "Class imbalance, redux," in *2011 IEEE 11th International Conference on Data Mining*, Dec 2011, pp. 754–763.

[15] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[16] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 559–563, Jan. 2017. [Online]. Available: http://dl.acm.org/citation.cfm?id=3122009.3122026

[17] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 233–240. [Online]. Available: http://doi.acm.org/10.1145/1143844.1143874