Investigating the optimum Data aggregation method for ANN based medical prediction algorithms

Musa Jabbar Taib
dept. of Electrical and Software Engineering
University of Calgary
Calgary, Canada
musa.taib@ucalgary.ca

Abstract — **Using machine learning to analyze patients' records and predict diseases has ushered us into a new age of healthcare. However, this new age has brought with it some new challenges as well. Meticulously detailed patient records are common in modern day, resulting in records varying greatly in size among patients in a hospital. This is problematic for machine learning application as most traditional machine learning algorithms expect fixed size datasets. To keep the data size consistent for different patients, this paper uses an interesting take on data aggregation, which involves converting data into 3 dimensional matrices. The principal goal of this paper is to find out the optimum number of windows that would give accurate results, and require as low on field computing power as possible. The results obtained support the principal goal, and imply some interesting real-world improvements to medical diagnosis applications.**

Keywords—Artificial neural network, machine learning, data science, medical diagnosis, data preprocessing.

I. Introduction

The advancement in computing technology has led to revolutionary developments in the domain of using machine learning to analyze data [1]. Artificial neural networks (ANNs) have seen a tremendous increase in their use because they are excellent at pattern recognition [2]. This ability to recognize patterns allows ANNs to predict the occurrence of a disease before it even happens. ANNs have been used in predicting diseases such as breast cancer, Hepatitis B, diabetes, and more [3]. This means that optimizing the on-field performance of an ANN would be quite beneficial to the society as it would mean faster and more accurate results.

However, the increased use of these machine learning models, particularly ANNs, has also highlighted a substantial weakness, which is the fact that most traditional machine learning models expect a constant sized dataset as their input. Medical data is often stored in a raw format making it very difficult to work on directly by a machine learning algorithm. The data has to pass through several layers of pre-processing such as data cleaning, data transformation, data aggregation, feature extraction, and feature selection, before it can be analyzed in a meaningful way.

The focus of this paper is a particular data pre-processing technique called data aggregation as, using data aggregation to pre-process medical data in machine learning applications has received considerable interest in recent years [4]. Data aggregation can be defined as summarizing large chunks of data into compact form in such a way that the accuracy of the insights from the data remains virtually the same. In [5], data aggregation was used for predicting diabetes early. While in [6], the authors divided their timeline into windows of 90 days, and aggregated the timeline to feed to several different machine learning models. A rule learning-based recommender system that could help homeless shelters in predicting chronic homelessness was developed using a rolling window mechanism in [7].

The contribution of this paper is that it studies the effect of using different sizes of windows (to aggregate medical records data) on a machine learning medical prediction model. Using different sizes of windows to determine if it impacts the prediction capabilities of a machine learning model has been done before like in [8], where 5 different window sizes were used and the difference in their prediction capabilities was studied. This paper focuses more towards determining if spending time on selecting an optimum window size in data aggregation reels enough benefits to warrant the time spent on it. The aim of this paper is to help reduce the computation power requirements of on-field medical diagnosis applications, while maintaining a high accuracy to promote higher adoption of these applications in resource constraint regions.

The rest of the paper is organized as follows. Description of the dataset, its creation and features are discussed in Section II. The pre-processing of the dataset and the window formation algorithm is discussed in section III part A. The neural network model and data computation using a high-performance cluster are explained in section III part B and C, respectively. Section IV presents the classification metrics used, followed by the discussion of the results in section V. Finally, the conclusion is made in section VI.

II. Data

A synthetically generated patients record dataset was used in this paper. It was made by pre-processing retro sheet event files from the Major League Baseball (MLB) dataset [9] to appear as a hospital records dataset [10]. MLB has several attributes that can be mapped to real world hospital data, making it ideal for this use case. For instance, if a player is released or traded from the roster, it can be considered as a patient getting released from a hospital which is mapped to the event of good test result. Likewise, many patients visit a hospital every day but only a few are in serious enough conditions that their vitals crash, and so the event of vitals crash has been attributed to the top players in the league. Top players are a small percentage of the total roster, and they stay the most active during the season, which is analogous to more hospital stays for those small number of patients who are in bad condition. The resultant is a multivariate time-stamped dataset, which is also how an official hospital records datasets should look like.

The synthetic dataset contains records for 915 patients over a period spanning from 1993 to 2015. It contains 452,995 rows and 3 columns titled player, date, and event in that order. The column titled player contains the name of the patient, while date contains the information regarding when the event occurred. The last column event contains the information that which of the four events (good test result, bad test result, stay and vitals crash) occurred during a particular visit. Vitals crash is the event that the ANN will be trained to predict. Vitals crash means that a patient's vital signs, which includes body temperature, respiration, and pulse rate, suddenly start deteriorating to a very dangerous extent. Conditions such as cardiac arrest can cause sudden vitals deterioration and predicting them beforehand can result in saving a patient's life.

III. Methodology:

A. Convert data into windows

For this part of the code, two functions were used: **timeline_summary** and **find_vitals_Crash** [11]. The first function goes through the entire timeline and counts the instances of occurrence of each of the four events in the hospital records dataset, and returns the aggregated data. This function also creates a new variable called tenure, that counts the number of days a patient stayed in the hospital. The latter function adds a column to the output of **timeline_summary** function which flags the patients who experienced a vitals crash.

Initially data from 1999 to 2013 (observation window) and 2014 (follow up window) are separately fed to **timeline_summary** and **find_vitals_crash** respectively. Since predicting results using less than 15 events during the obersvation window would cause very inconsistent results. So, all such patients (6 total) were removed from the dataset in the next step.

Then the observation window was divided into n windows using the variable number of windows and each window was sent to the **timeline_summary** function. To ensure that the resultant dataset had the same number of columns for each patient, if a patient did not have any event in a window, zero padding was used. The resultant dataset from the previous stage was fed into a function called **generate_sequences**. This function converted the input dataframe to a 3d NumPy array of size [(Number of Windows) x (Number of Events) x (Number of Patients)].

B. Predicting Vitals Crash

Data pre-processing stage gave two arrays, one 3d array for the observation window and one 1d array with flag columns for the year 2014. Stratified k-fold cross-validation was used as the data re-sampling technique to mitigate the imbalanced nature of the original dataset during testing, to assess the generalized capabilities and to prevent over-fitting [12]. In stratified k fold cross validation, roughly the same proportion of the two flags i-e vitals crash and not vitals crash in this case, is contained per fold as in the original dataset during testing.

An ANN was used in this paper to predict the possibility of a patient experiencing vitals crash in 2014. The neural network had 4 layers with an input flatten layer, followed by 3 dense layers with 10, 16, and 1 units respectively. Because of the binary nature of the classification, sigmoid was used as the activation function, and binary cross entropy was used to calculate loss. Lastly, adam was used as the optimizer.

The ANN used in this research is a relatively standard ANN design. This model could have been much more optimized by fine tuning the hyper-parameters as discussed in [13]. However, that route is not explored here, as the focus of this paper was to study the effect of changing the window size on the performance.

C. High Performance Cluster

As discussed during the introduction, the optimum window size while computationally less taxing, should also perform well in term of predicting results. The University of Calgary's Teaching and Learning Cluster (TALC) which is a high-performance cluster, was used to compute all the results quickly for process of filtering out the window sizes that perform well in terms of predicting results.

The operating system of TALC is simple linux utility for resource management (SLURM) [14], which is cluster management and job scheduling system and so separate codes had to be written to submit jobs to it. Special care was taken to write the python codes in such a way that they would be considered as embarrassingly parallel codes [15]. This allowed several hundred instances of the code to be run in parallel using the SLURM job array mechanism [16]. The results of the classification matrices were stored in a CSV file, which was then used to examine the results produced.

IV. Classification Metrics

Before discussing the metrics used, some terms need to be explained for better understanding. True positive (TP) is the case where a patient had a vitals crash, and the model also predicted a vitals crash. True negative (TN) is the case where a patient did not experience a vitals crash, and the model also predicts the patient to not experience a vitals crash. False positive (FP) is the case where a patient had not experienced a vitals crash, and the model predicted a vitals crash event. False negative (FN) is the case where a patient had a vitals crash, but the model predicted that the patient will not experience a vitals crash.

A. Accuracy (A)

Accuracy is used to judge how good an algorithm is overall at predicting the correct answer, and is generally adequate at gauging the performance of a model. However, in this case, using accuracy alone is not desirable because of the imbalanced nature of the data (866 out of the total 915 patients don't experience vitals crash). Even if the model predicted all 915 patients to not experience a vitals crash, the accuracy would still be above 94%

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} .$$

B. Precision (P)

Considering the event of a patient experiencing vitals crash as the target event, we can define precision as

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} .$$

Precision for this event gives the probability that when the model predicted that a patient would experience a vitals crash, how often did they really experience a vitals crash. This means a model with high precision is better for this use case.

C. Recall (R)

Considering the event of a patient experiencing Vitals crash as the target event, we can define recall as

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} .$$

Recall for this event gives the probability that when a Patient was actually about to experience a vitals crash, how often did the model predict they would experience it. This means a model with high recall is better for this use case.

D. F1 Score (F1_Score)

A high F1 score means both a high precision and a high recall as it the harmonic mean of the two terms. For this paper, the F1 Score for the event of experiencing a vitals crash is labelled as F1_Score_0. While F1_Score_1 is the label used for the event of not experiencing a vitals crash

$$\text{F1 Score} = \frac{2*(\text{P} * \text{R})}{\text{P} + \text{R}} .$$

E. Macro Average of F1 Scores (MA)

A high F1 score for predicting both the events is equally important as more lives will be saved by predicting more accurately the patients who will or will not experience vital crashes. If hospitals know which patients to prioritize, they can make an optimal allocation of scarce health care resources. The category of macro average takes an average of both the F1 scores, meaning it shows which window size not only performs better overall, but it is equally competent at predicting both patients who will, and who will not experience vitals crash soon

$$\text{Macro Average} = \frac{\text{F1 Score 0} + \text{F1 Score 1}}{2} .$$

V. Simulation Results

A. Optimum Window Size

It can be observed from figure 1 & 2 that the imbalance nature of the dataset shows itself in the results obtained. Meaning that for a majority of cases both the accuracy and F1 Score 1 remain above the 90% threshold, even though figure 2 clearly shows that the model was poor at predicting patients with vitals crash most of the time. This means that accuracy and F1 Score 1 alone are not good parameters to base a decision on.

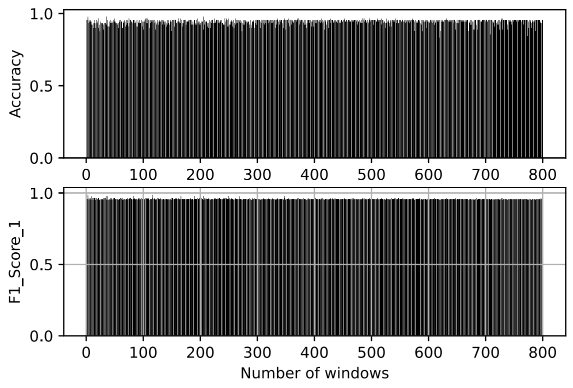

Figure 1: Accuracy & F1 Score 1 [Bar Charts]

The values of F1 Score 0 and macro average meanwhile change drastically as clear from figure 2. From figure 2, it is also clear that macro average is the foremost metric to determine the performance of different number of windows, as it gives equal weightage to predicting both the classes , which is evident from the much higher average reading as compared to F1 Score 0.

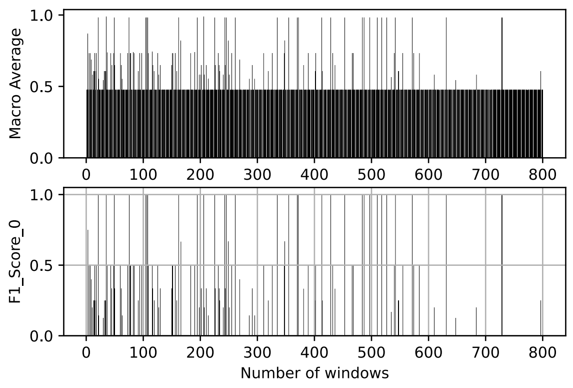

Figure 2: Macro average & F1 Score 0 [Bar Charts]

The top 5 results for each metric have been compiled in figure 3. It can be observed from figure 3 that dividing the timeline into 35 or 206 windows have the highest macro average, meaning that they performed the best. These two window sizes also give some of the best accuracies, meaning selecting window sizes based on macro average of the F1 scores also gives accurate results, as it obviously should.

	A	*F1_0*	*F1_1*	*MA*
1	0.9778/3	0.9884/3	1/225	**0.9833/35**
2	0.9778/35	0.9881/116	1/245	**0.9833/206**
3	0.9778/206	0.9773/35	1/35	**0.9752/49**
4	0.9667/261	0.9773/206	1/370	**0.9752/527**
5	0.9667/249	0.9770/348	1/206	**0.9752/484**

Figure 3: Table of top 5 values of all 4 metrics

B. *Real-world Computation time*

Another very important aspect of a machine learning model that is to be implemented on-field is the time it takes for the model to train. To calculate the computation time for the on-field implementation, the code had to be rerun on a laptop. Training the model for 800 window sizes took 8 hours to compute on TALC. However, the same task on an average laptop would have taken over 16 days. Therefore, from figure 4, it can be observed that 17 equally spaced window sizes were taken, and their computation time was plotted in figure 4. All other compute times were easily extrapolated from the graph due to its linear nature.

From these figures, it can clearly be concluded that, even though keeping the number of windows equal to 36 or 205, both performed equally well in the previous matrices. For computation time, using 36 windows reduces the time by almost 5 times or, in other words reduces the computation power requirement of the machines running these algorithms by 5 times. This result means that spending time on select the optimum window size is worth the resources, considering the net time saved during testing and recalibration

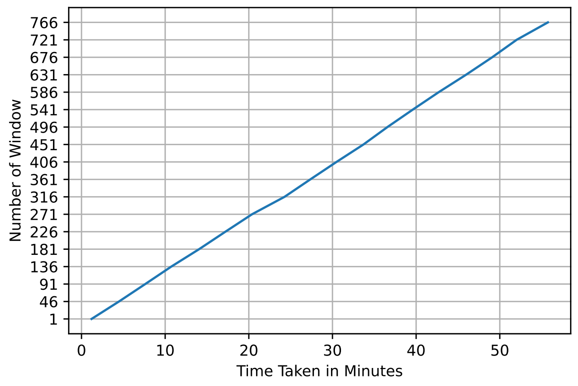

Figure 4: Computation time required Vs Number of windows

VI. Conclusion

Optimizing the on-field performance of diagnostic machine learning applications to make them more reliable, faster, and accessible to others, has always been the goal of engineers and data scientists. In recent times, more and more machine learning algorithms have been transitioning into real-world front-line applications to help doctors and patients alike. This means optimizing these algorithms will result in many benefits to the society, from better allocation of already scarce medical resources in 3rd world countries, to diagnosing life-threatening diseases in time.

This paper aims to play a small part in improving the field of data science, by demonstrating that carefully selecting the size of the data aggregation window can increase the performance of diagnosis application by several folds. This also means that by keeping the computation speed the same, the processing power requirement of these diagnosis applications can be reduced by an appreciable amount, reducing the overall computation power requirements/cost of these applications. A reduction in cost would make these applications accessible to a wider population, and help more lives.

While this paper limited itself only to generic ANN models. It would be an interesting future work to study these effects on a wider range of machine learning algorithms like recurrent neural networks, convolutional neural networks, k nearest neighbors and many more.

References

[1] Sarker, I. H. (2021). Machine learning: Algorithms, real-world applications and Research Directions. SN Computer Science, 2(3). https://doi.org/10.1007/s42979-021-00592-x

[2] Abiodun, O. I., Kiru, M. U., Jantan, A., Omolara, A. E., Dada, K. V., Umar, A. M., Linus, O. U., Arshad, H., Kazaure, A. A., &amp; Gana, U. (2019). Comprehensive Review of Artificial Neural Network applications to pattern recognition. IEEE Access, 7, 158820–158846. https://doi.org/10.1109/access.2019.2945545

[3] TRIPATHI, K. S. H. I. T. I. J. (2019). Diabetes classification and prediction using Artificial Neural Network. INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING &amp; TECHNOLOGY, 10(3). https://doi.org/10.34218/ijcet.10.3.2019.018

[4] Rolnick, J. (2013). Aggregate Health Data in the United States: Steps toward a public good. Health Informatics Journal, 19(2), 137–151. https://doi.org/10.1177/1460458212462077

[5] Mahboob Alam, T., Iqbal, M. A., Ali, Y., Wahab, A., Ijaz, S., Imtiaz Baig, T., Hussain, A., Malik, M. A., Raza, M. M., Ibrar, S., & Abbas, Z. (2019). A model for early prediction of diabetes. Informatics in Medicine Unlocked, 16, 100204. https://doi.org/10.1016/j.imu.2019.100204

[6] Messier, G., John, C., &amp; Malik, A. (2021). Predicting chronic homelessness: The importance of comparing algorithms using client histories. Journal of Technology in

Human Services, 1–12. https://doi.org/10.1080/15228835.2021.1972502

[7] C. T. John, Design, and Implementation of a Recommender System for use at an Emergency Homeless Shelter in Calgary, MSc. thesis, Dept. Elect. and Comp. Eng , Univ. of Calgary, Calgary, Canada, 2021

[8] Messier, G., Tutty, L., &amp; John, C. (2021). The best thresholds for rapid identification of episodic and chronic homeless shelter use. International Journal on Homelessness, 1–11. https://doi.org/10.5206/ijoh.2022.1.13607

[9] Messier, G. G. (n.d.). Data-analytics/create MLB hospital data set.ipynb at D9AC085453A21FE65F6464FC67B24F49707BE183 · GGMESSIER/data-analytics. GitHub. Retrieved December 14, 2021, from https://github.com/ggmessier/data-analytics/blob/d9ac085453a21fe65f6464fc67b24f49707be183/baseball/Create%20MLB%20Hospital%20Data%20Set.ipynb.

[10] MLB Retrosheets. Retrosheet event files. (n.d.). Retrieved December 14, 2021, from https://www.retrosheet.org/eventfile.htm.

[11] Messier, G. G. (n.d.). Data-analytics/exploratory data analysis.ipynb at D9AC085453A21FE65F6464FC67B24F49707BE183 · GGMESSIER/data-analytics. GitHub. Retrieved December 14, 2021, from https://github.com/ggmessier/data-analytics/blob/d9ac085453a21fe65f6464fc67b24f49707be183/demo/Exploratory%20Data%20Analysis.ipynb.

[12] Hastie, T., Tibshirani, R., &amp; Friedman, J. (2008). Neural networks. The Elements of Statistical Learning, 389–416. https://doi.org/10.1007/978-0-387-84858-7_11

[13] Yu, Tong, and Hong Zhu. “Hyper-Parameter Optimization: A Review of Algorithms and Applications,” 2020.

[14] https://slurm.schedmd.com/tutorials.html

[15] freeCodeCamp.org. (2021, April 28). Embarrassingly parallel algorithms explained. freeCodeCamp.org. Retrieved December 14, 2021, from https://www.freecodecamp.org/news/embarrassingly-parallel-algorithms-explained-with-examples/.

[16] Slurm Job Array. Slurm Workload Manager - Job Array Support. (n.d.). Retrieved December 14, 2021, from https://slurm.schedmd.com/job_array.html.