Big Data for Health: Reproducibility Challenge

Siddiqui, Muhammad N, Pratt, Luke O
Georgia Institute of Technology
msiddiqui43 903644474, lpratt30 903265437
git-repository https://github.gatech.edu/lpratt30/PredictingHF/tree/dev
presentation https://drive.google.com/file/d/1193f0dH4bQVdGwLypwOTTqYY1wHeZX2y/view

1. Introduction

We test the reproducibility of the paper "*Predicting Heart Failure Readmission from Clinical Notes Using Deep Learning*" [4]. The paper predicts general heart failure readmission and 30-day readmission using patient discharge reports from the MIMIC-III v1.4 database, a set of de-identified medical data from 46,520 patients from critical care units at Beth Israel Deaconess Medical Center [2].

Their model is a convolutional neural network (CNN) trained on word embeddings of words from the discharge reports. Their hypothesis is that a CNN is a competitive model in healthcare for predictive analysis when incorporating unstructured natural language.

Language data is common yet challenging. The benefits are that this model can be created rapidly with no domain expertise. This is in contrast to language models requiring expert hand-crafted features, the prior state of the art in the field at the time of publication. They achieve high F1 scores of 0.756 and 0.733 for general and 30-day readmission.

They baseline on random forest (RF), where TF-IDF (term frequency inverse document frequency) form features. Random forest with TF-IDF is a popular model for language based predictions and also does not need feature engineering. The RF achieved F1 scores of 0.674 and 0.656 respectively, outperformed by the CNN model.

They then propose a chi-squared feature analysis for interpretability as to what their model bases its predictions from. The chi-square score of words from correct predictions are ranked, giving insight to their predictive relevancy.

Our findings discovered challenges in reproducibility and do not support the hypothesis. We achieved better performance than the CNN with random forest. We did not achieve the same CNN performance after tuning and referencing the architecture source [3]. We also found the use of balanced data for reported performance was understated.

2. Scope of reproducibility

We reproduce their dataset and its statistics, their primary models (random forest and CNN), and the chi-squared feature analysis. We create the accompanying visualizations for each of these components.

We diverge where we don't have the full details of their data processing or model hyperparameters, opting to generally follow sound data and model handling practices. Additionally, we train in random batches from the set of shuffled training data rather than using 10-fold cross-validation.

3. Methodology

3.1. Hardware

We use personal workstations and GPU primarily. We have RTX 4000 and RTX 3080 across our team members, either is able to handle the workloads. The workstations respectively have 64 and 32GB of physical memory. The max wall time training time we experience for a training epoch with any of the models (i.e., random forest with hundreds of estimators and 25,000 features) is around 15 minutes.

3.2. Code

All code is written from scratch as the paper's was not public. We primarily use Scikit-learn as our source of models (they also state using this in the paper), as well as PyTorch, NumPy, and Pandas for data processing.

We also write the query to collect the data via SQL with Google Big Query (free to use in this case). Without Google Big Query, the 1TB MIMIC-III database would need to be downloaded and filtered locally.

Our code and instructions to run it can be accessed here. If you cannot access it, please email us:

- https://github.gatech.edu/lpratt30/PredictingHF/tree/dev

Because the github repository had to be made public to share, but not this document, we provide the access to the dataset referred to in the readme with this password: "BD4HfinalProj".

3.3. Data collection

The ICD-9 (international classification of diseases) codes mentioned in the paper were used to query hospital admissions for any admissions that were heart failure related. This diagnosis accompanies a hospital admission ID where it was received. Then, the hospital admissions are queried for those hopsital admission IDs for related data such as the discharge reports and patient identifiers.

To match the dataset statistics of the original study, we had to deduce certain logic in writing the SQL code. Specifically, we discovered that the original study counted the 30th day as a full 24-hour period for readmission logic, effectively 31 days, rather than capping it at the end of the 29th day. This discrepancy easily arises depending on the data querying tools and services used, such as Google Big Query, Amazon Web Services, or Python functions. Additionally, with minor impact on the dataset's statistics, we found that they had excluded 'addendum' entries from the TEXT data category, focusing solely on 'report' categories.

We subsequently loaded the data pertaining to heart failure patients into a GBQ table. This table included: admission id, subject id, and clinical notes, as well as a target column for readmission and another for 30-day readmission that were determined with GBQ DATEDIFF function. Following this, we exported the table to a CSV file and load the CSV as a Pandas dataframe locally.

As shown below, statistics matching their dataset were able to be reproduced in Google Big Query. It is possible this isn't the *exact* same data, but unlikely.

Table 1: Reproduced dataset statistics

	# Admissions	# with notes
All admissions	14,040	13,746
Readmissions	3,604	3,543
30-day readmissions	969	962

As in the paper, two datasets are created from the above samples. The data is also balanced. This implies that the each dataset is from a stratified set of positive and negative samples of the respective readmission case. The prediction problem is simplified because the predictions aren't being drawn from a real and imbalanced population. This is mentioned in the paper's discussion on data processing.

3.4. Data processing

Next, the data is processed to remove stop words, punctuation, and numbers, before tokenizing. The CNN is trained on word2vec vectorized contextual meanings of words. The same PubMed-and-PMC-word2vec embeddings [5] mentioned in the paper with a dimensionality of 200 is used. Dimension here refers to contexts of association for the word [7].

While analyzing the processed data, we noticed that the count of words per record is right-skewed with high kurtosis, as shown in Fig. 1. Had we chosen the maximum word count as the first dimension of our embedding vector, it would have placed an unnecessary burden on the system. Moreover, this approach would have resulted in excessive zero padding, potentially impacting the modeling results. Therefore, we opted to use the interquartile range between the 85th and 25th percentiles as our cutoff value. Since CNNs require datasets of uniform length, we applied zero padding to records with fewer words up to this interquartile cutoff. For records exceeding this limit, any words beyond the cutoff were disregarded.

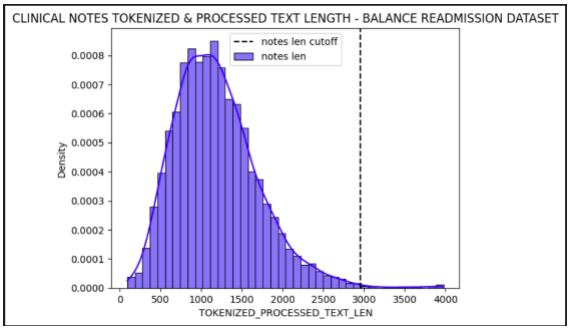

Figure 1: Word Count Distribution of Processed and Tokenized Notes.

We then created a balanced dataset of heart failure patients, comprising equal numbers of readmitted and non-readmitted individuals, resulting in 3,543 patients in each category. We split the data into training and test sets using a 90 to 10 ratio.

Initially, we planned to transform the data and extract embeddings from the pre-trained model in a single step. However, this approach quickly proved to be impractical, as it required more memory than the 64GB available on our local machines. Faced with this challenge, we explored alternative methods to obtain embeddings without overloading our system's resources.

Our solution leveraged the capabilities of PyTorch, a framework that offers specialized classes for efficient data handling. In PyTorch, we had two primary options for processing our data: we could either process each individual record using a custom implementation of the Dataset class or process entire batches using the collate_fn in the DataLoader class. After careful consideration, we opted for the latter approach.

Implementing a custom collate_fn function allowed us to process and obtain embeddings for entire batches, significantly reducing the memory footprint. This batch process-

ing approach not only aligned with the batch optimization of our pre-trained model, enhancing performance and efficiency, but also offered scalability and optimal resource utilization. It proved to be a versatile and robust solution, enabling us to train our model effectively without hardware upgrades. This method resolved our memory issues and provided a blueprint for efficient data handling in future machine learning tasks.

3.5. Training random forest

We first create the baseline model. The critical step was to get TF-IDF (Term Frequency - Inverse Document Frequency) [6]. It is a logarithmic ratio of many documents there are relative to the frequency of documents containing the term, weighted by the number of documents containing the term.

$$TF = \frac{\text{count of term in the document}}{\text{number of terms in the document}}$$

$$IDF = \log\left(\frac{\text{number of documents}}{\text{number of documents containing the term}}\right)$$

$$TF\text{-}IDF = TF \times IDF$$

Here, "documents" are the hospital discharge report, or entries in the Pandas dataframe. Sklearn handles the transformation of the dataframe with its Tfidvectorizer. The vectorizer is fit only to the training corpus to avoid being able to model from test data. When fitting the transformation to the test data, Tfidvectorizer simply omits words it has no recorded entries for.

With respect to the classification problem and using TF-IDF scores as decision points in decision trees, it can be thought of as determining what populations of terms may result in heart failure readmission, agnostic to their locality. In a random forest, that goes one step further to considering many different compositions of possible terms and voting upon them. The lack of locality is opposed to CNN, where feature locality (positional relationship of a word in a sentence) is learned by the kernel size and weights.

In the original paper, they explain that they tuned a random forest with between 10,000 and 25,000 features at intervals of 5,000. However, random forest has a critical parameter being the number of trees created, and they didn't state the number of trees they used. We applied grid search with numbers of estimators and varying maximum features for each estimator.

3.6. Training CNN

In our text analysis using CNNs with 2D matrix vector embeddings of words, we drew upon the foundational work of Yoon Kim in 'Convolutional Neural Networks for Sentence Classification' [3], which demonstrated the effectiveness of CNNs in NLP tasks. We trained the CNN model using embedding vectors, adapting the approach where Kim's paper did not specify hyper-parameters. It detailed only the architecture, mentioning the use of various filters with kernel sizes of 1, 2, and 3, and the employment of a ReLU activation function between filters, followed by global max pooling and a SoftMax function.

In our implementation, we utilized three convolutional layers with these kernel sizes, setting the output channels, or neurons, for each convolutional layer to 100. This approach allows the network to employ convolutional filters of varying sizes to extract meaningful patterns from sentences, where each word is represented as a vector in a high-dimensional space, forming a 2D matrix. Filters of different sizes slide over this matrix, capturing various aspects of the data: single-row filters analyze individual words, while larger filters spanning multiple rows detect patterns in word sequences, akin to bi-grams or tri-grams. Through feature maps and pooling layers, the network distills these patterns into salient features, enabling it to understand and interpret the semantic relationships and context within the text. This hierarchical feature extraction, as demonstrated in Kim's work, makes CNNs effective for a range of NLP tasks, as they can capture and utilize patterns at different levels of complexity, from single words to phrases

After filtering, we applied a Rectified Linear Unit (ReLU) activation function, introducing non-linearity to the network and enabling the model to learn complex patterns. We then employed global max pooling to reduce the size of the feature maps by taking the maximum value across each map, thus distilling the most crucial information from each feature map.

To prevent overfitting, we incorporated a dropout layer that randomly omits some information during training. Conceptually, this transforms the neural network into an ensemble learner [1]. The features extracted and refined by the convolutional and pooling layers are then fed into a fully connected layer, integrating these features to form a final prediction. The output from this layer determines the predicted class or value of our model's target.

Table 2: CNN Hyper Parameters

Task	Batch	Filter Size	# of Filters	LR	Dropout
GR	32	[2,3,4,5]*2	[200]*8	0.0001	0.3
30Day	32	[1,2,3,4]	[200]*4	0.001	0.1

3.7. Chi-squared analysis

Interpreting the results of deep learning models, such as Convolutional Neural Networks (CNNs), can be challeng-

ing. To aid in this interpretation, the authors proposed using chi-square analysis for the prediction results of CNN models. The chi-square test, a statistical method, is employed to determine if there is a significant association between two categorical variables.

In the context of CNNs, chi-square analysis is particularly useful for analyzing or interpreting the importance or relevance of different input features, such as words in clinical notes, in relation to the model's predictions. By integrating chi-square analysis, we can statistically validate which features are not only being activated but are also statistically significant in relation to the output.

Our process begins by categorizing the predictions made by our CNN. We separate the samples into those that were correctly predicted, where the predicted class matches the actual class, and optionally, those that were incorrectly predicted for a more comprehensive analysis.

Next, we conduct a chi-square test for each feature, such as words in text data. This involves several key steps. First, we calculate the frequency of each feature's occurrence in each class, be it positive or negative. To perform the chi-square test, we first need to calculate the total occurrences of each word. This is done by summing the observed frequencies of the word in both positive and negative classes:

$$\text{Total occurrences} = \text{Positive count} + \text{Negative count}$$

Then, we calculate the proportion of words in each category. This is the ratio of the total number of words in a category (positive or negative) to the total number of words in both categories:

$$\text{Positive word ratio} = \frac{\text{Total positive words}}{\text{Total words}}$$

$$\text{Negative word ratio} = \frac{\text{Total negative words}}{\text{Total words}}$$

Subsequently, we determine the expected frequencies. The expected frequency of the word in each class is calculated as the total occurrences of the word multiplied by the proportion of words in that class:

$$\text{Expectation positive} = \text{Count} \times \text{Positive word ratio}$$

$$\text{Expectation negative} = \text{Count} \times \text{Negative word ratios}$$

The chi-square test then compares these observed frequencies (O_i) with the expected frequencies (E_i), calculated as above. The chi-square formula is given by:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

where O_i is the observed frequency, and E_i is the expected frequency. This calculation helps us determine the statistical significance of the association between features and classes.

4. Results

The below heatmap shows the F1 scores for the tuning of random forest on general readmission:

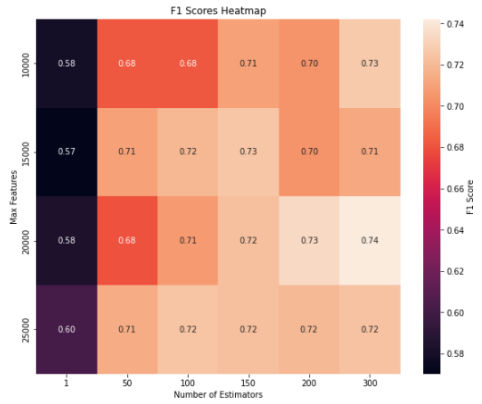

Figure 2: Random Forest tuning for general readmission.

Shown below are the CNN training curves, truncated at saturation:

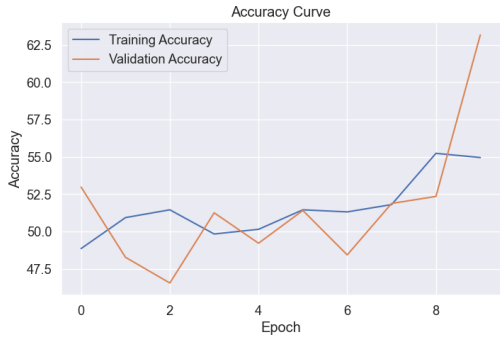

Figure 3: CNN general readmission Accuracy curves.

Figure 4: CNN general readmission Loss curves.

Table 3 shows how the models performed in the original study, while Table 4 shows the performance results of the reproduced models:

Table 3: Paper's performances

Task	Model	F1	Acc
General readmission	CNN	0.756	75.70%
	RF	0.674	69.35%
30-day readmission	CNN	0.733	71.88%
	RF	0.656	67.19%

Table 4: Reproduction's performances

Task	Model	F1	Acc
General readmission	CNN	0.625	60%
	RF	0.7421	72.4%
30-day readmission	CNN	0.629	62.1%
	RF	0.7265	65.28%

Although we replicated other aspects of the paper accurately, including dataset statistics and superior Random Forest performance compared to the original paper, we consistently obtained subpar results for the CNN model. This suggests that our interpretation and implementation of the CNN architecture might have been off the mark, or that the original paper may have omitted critical information.

In some 30 day readmission training trials as shown in the appendix, we saw random forest F1 scores as high as 0.8. It was found that the highest performance came when stratifying on the already balanced dataset. The 30 day performance is sensitive to which samples are selected, as the test set for 30 day readmission (10% of 962 samples) only has 96 items.

Figure 5 displays the top 30 key features based on the highest to lowest chi-square values for both the General Readmission CNN. The features are listed along the vertical axis, and the length of each bar represents the chi-square value for that feature. The number at the tip of each bar indicates the frequency of that feature's appearance, and the letter next to it denotes the dominant class.

While we were unable to replicate the original paper's performance for the CNN model, we did observe some overlapping features (highlighted in yellow) that also appeared in the chi-square results of the original study.

5. Discussion

This paper has mixed issues in terms of reproducibility. For the hypothesis that CNN on word2vec is a powerful model superseding hand-crafted features to be replicated, our CNN must have better relative performance to the random forest. Our random forest performs markedly better

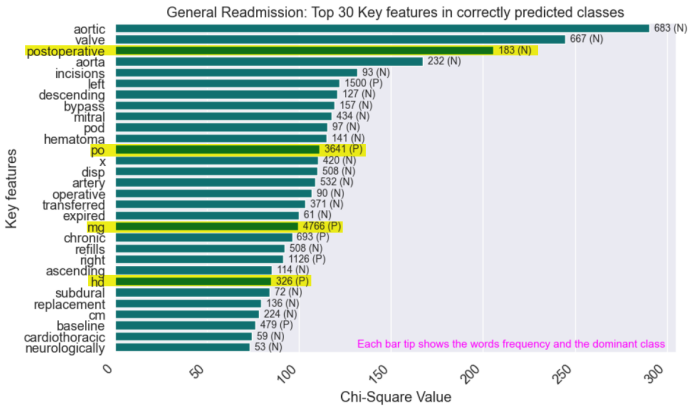

Figure 5: Key Features in correctly predicted samples for general readmission.

than the paper's random forest.

One point of notice is all of model evaluations and training are done on a balanced dataset. In reality, general heart-failure readmission is imbalanced, and 30-day moreso. The implications of a balanced dataset were understated in the original paper. For example, the abstract reports the achieved performances in a generalized sense without mentioning the use of the balanced datasets, and it isn't mentioned in introduction or conclusion. While this setup allows comparison of models, it could be misleading at a high level for the capabilities of these models, especially when using F1 score as the primary metric.

The paper did an excellent job of explaining the high level process, but not enough details that their work could readily be recreated. The most challenging aspects of this work were in dealing with the many possible variations in data processing, tuning methodology, and missing hyperparameters, while trying to exactly replicate performance.

Full determination of the CNN's and RF's performance discrepancy is not possible without the original code. The lack of code puts the onus on us as reproducers to match the performance of the models. When we have a discrepancy such as much better performance than the paper's results with random forest and lower performance with CNN, it is impossible for us to know what steps we may take to improve the model that may have not been stated, or what specifically caused us to achieve better baseline.

We conclude that it is difficult to *exactly* convey the methodology in data intensive research while balancing the brevity of a publication. We recommend that code is included and randomness set to a seed, and that hyperparameters are systematically detailed. We appreciate the paper's effort to flow well at a high level and be concise, but an additional table of hyperparamters or provided code would have solved many issues.

References

[1] Kazuyuki Hara, Daisuke Saitoh, and Hayaru Shouno. Analysis of dropout learning regarded as ensemble learning. In *Artificial Neural Networks and Machine Learning–ICANN 2016: 25th International Conference on Artificial Neural Networks, Barcelona, Spain, September 6-9, 2016, Proceedings, Part II 25*, pages 72–79. Springer, 2016. 3

[2] Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016. 1

[3] Yoon Kim. Convolutional neural networks for sentence classification. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. 1, 3

[4] Xiong Liu, Yu Chen, Jay Bae, Hu Li, Joseph Johnston, and Todd Sanger. Predicting heart failure readmission from clinical notes using deep learning. In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 2642–2648. IEEE, 2019. 1

[5] SPFGH Moen and Tapio Salakoski2 Sophia Ananiadou. Distributional semantics resources for biomedical text processing. *Proceedings of LBM*, pages 39–44, 2013. 2

[6] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 29–48. Citeseer, 2003. 3

[7] Hinrich Schütze. Dimensions of meaning. *SC*, 92:787–796, 1992. 2

A. Appendix

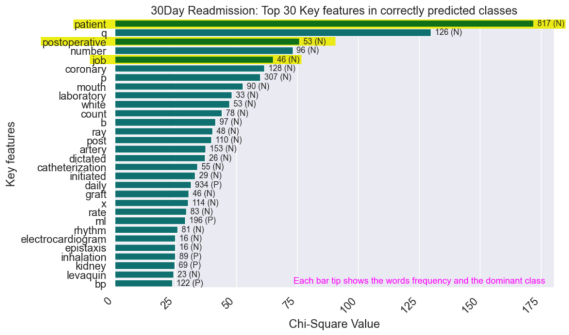

Figure 11: Key Features in correctly predicted samples for 30day readmission.

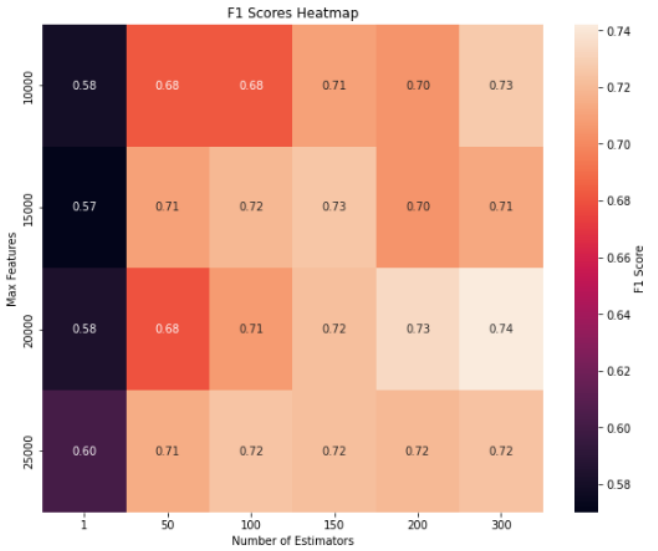

Figure 6: Random Forest tuning for 30 day readmission.

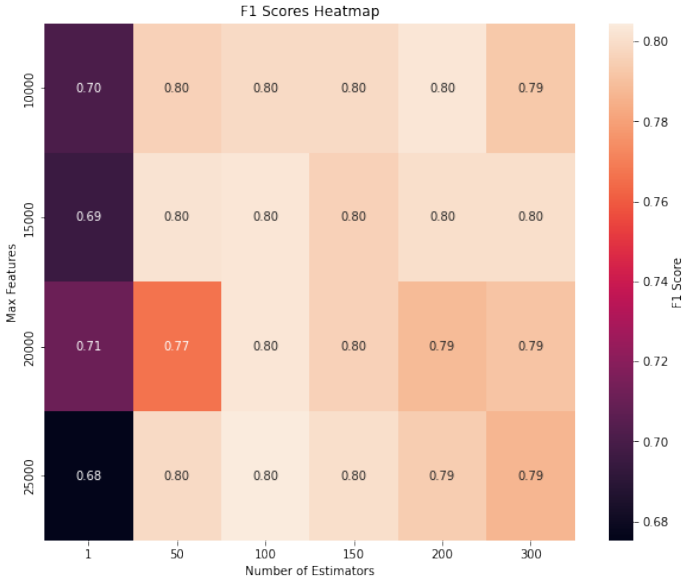

Figure 7: Random Forest tuning for 30 day readmission, exceptional trial.

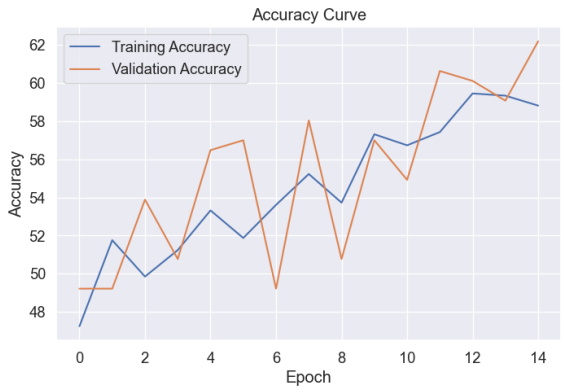

Figure 8: CNN 30day readmission Accuracy curves.

Figure 9: CNN 30day readmission Loss curves.

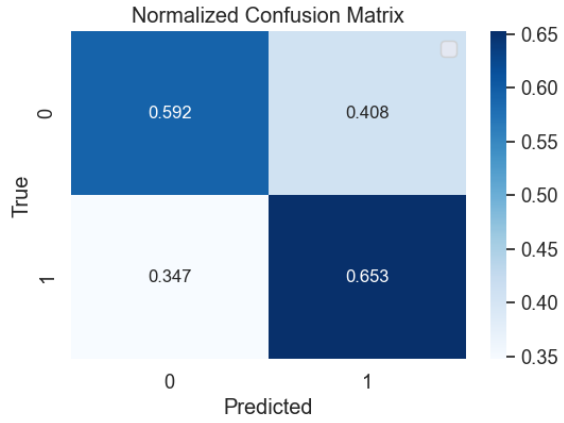

Figure 10: CNN 30day readmission Confusion matrix.

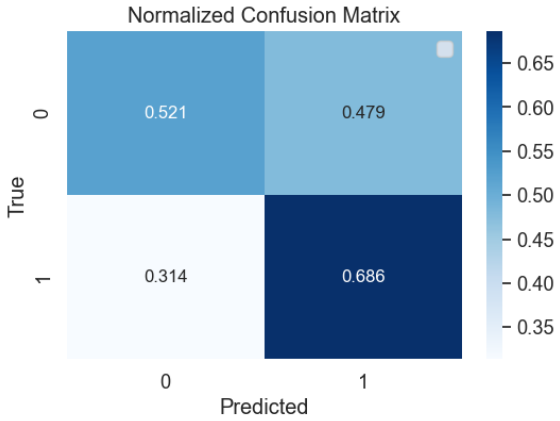

Figure 12: CNN general readmission Confusion matrix.