Note: To properly view this report, please open it with Microsoft Word, as other software may result in formatting issues.

Group Members:

Pu Rui Ling 3035917989

Jiang Xiao 3035951894

Lyu Zhi Heng 3035772432

Wong Wai Chak 3035784186

CHAN Hiu Yu 3035784162

**STAT3612 Group Project Report**

**2023-2024 Semester 1**

**ABSTRACT**

The evaluation of traditional machine learning models and deep learning models on smaller datasets has been a recurring subject of debate. Despite the rising interest in the application of deep learning methods, traditional machine learning models, particularly those based on decision trees, continue to be a steadfast choice for such challenges. In this study, we aimed to further investigate this issue by using the 30-day hospital readmission rate as our target. We played an extensive assortment of 10 different state-of-the-art models, with a comprehensive analysis of their strengths and weaknesses.

Table of Contents

[Section 1: Data & Preparation 3](#_Toc152843693)

[1.1. Data Analysis 3](#_Toc152843694)

[1.2. Feature Selection: Mallows’ Cp Statistic and Correlation Analysis 4](#_Toc152843695)

[1.3. Feature Engineering 4](#_Toc152843696)

[Section 2: Parameter Adjustment 6](#_Toc152843697)

[2.1. Searching Algorithms 6](#_Toc152843698)

[2.2. Model Training and Evaluation 7](#_Toc152843699)

[Section 3: Machine Learning Models 8](#_Toc152843700)

[3.1 Traditional Machine Learning Algorithm 8](#_Toc152843701)

[**Section 4: Deep Learning Models 10**](#_Toc152843702)

[4.1. MLP 10](#_Toc152843703)

[4.1.1. Method 10](#_Toc152843704)

[4.1.2. Observation and Result 11](#_Toc152843705)

[4.2. LSTM 11](#_Toc152843706)

[4.2.1. Method 11](#_Toc152843707)

[4.2.2. Observation 12](#_Toc152843708)

[4.2.3. Result 12](#_Toc152843709)

[4.3. Fusion Network 13](#_Toc152843710)

[4.3.1. Method 13](#_Toc152843711)

[4.3.2. Observation 13](#_Toc152843712)

[4.3.3. Results 14](#_Toc152843713)

[4.4. Metric Learning 14](#_Toc152843714)

[4.4.1. Model Architecture 14](#_Toc152843715)

[4.4.2. Loss Function 15](#_Toc152843716)

[4.4.3. Result 15](#_Toc152843717)

[4.5. GNN 15](#_Toc152843718)

[4.5.1. Graph Construction 15](#_Toc152843719)

[4.5.2. Model Architecture 16](#_Toc152843720)

[4.5.3. Observation 16](#_Toc152843721)

[4.5.4. Result 16](#_Toc152843722)

[Section 5. Discussion 17](#_Toc152843723)

[Section 6. Conclusion 18](#_Toc152843724)

[Appendix A 18](#_Toc152843725)

Section 1: Data & Preparation

In the initial stage of our project, we performed a comprehensive analysis of the original dataset, which consisted of 171 different features. These features encompassed demographics data, comorbidities recorded as ICD-10 codes, lab test results, and medication data.

* 1. Data Analysis

Key metrics such as mean of age, gender distribution, and length of hospital stay have been calculated. Additionally, we identified the top five most common diseases based on their occurrence rates in the dataset. Our findings indicated that the average age was slightly higher, while the gender ratio was relatively balanced.

|  |  |  |
| --- | --- | --- |
| Age (Mean) | 66.23 | |
| Gender (%) | Male | 6337（54.65%) |
| Female | 5259（45.35%) |
| Length of stay (Mean) | 15.09 days | |

Figure 1. Key metrics of data

|  |  |  |
| --- | --- | --- |
| Comorbidity (Count) | I: Infectious and Parasitic Diseases | 1630 |
| R: Diseases of the Circulatory System | 747 |
| D: Diseases of the Digestive System | 603 |
| E: Endocrine, Nutritional, and Metabolic Diseases | 436 |
| J: Diseases of the Respiratory System | 409 |

Moreover, we recognized the importance of comorbidities, where patients have multiple concurrent diseases, in impacting the risk of readmission. Comorbidities can complicate patients' overall health status, increase treatment plan complexity, and potentially lead to readmissions. We believe a thorough exploration and understanding of these features could potentially help for our feature engineer section.

1.2. Feature Selection: Mallows’ Cp Statistic and Correlation

We first tried two simple feature selection metrics: Mallows’ Cp statistic and correlation analysis. Mallows’ Cp statistic can explore the trade-off between model complexity and fit. We identified the most important features by ranking them based on their Cp values, with lower Cp values indicating higher significance. On the other hand, Correlation analysis could identify features with higher potential importance with the target variable. The absolute correlation between all the features and the readmission rate was calculated. Various correlation thresholds (0.7/0.8/0.9) were explored,

However, neither of the features selection metrics yielded improved ROC-AUC performance. As a result, we decided to retain all 171 features in our models for further evaluation and analysis. We believe the tree-based method can do feature selection themself, and regularizations could be added for other machine learning models (regression models).

Despite our rigorous feature selection attempts using Mallows’ Cp statistic and correlation analysis, neither method led to improved ROC-AUC performance. Consequently, we chose to incorporate all 171 features in our models for subsequent evaluation and analysis. We posit that the tree-based method inherently performs feature selection, reducing the necessity for explicit pruning. We also suggest the use of regularizations in other machine learning models, such as regression models, to manage feature complexity.

* 1. Feature Engineering

As shown in Figure 2, different feature engineer strategies were explored to enhance our models based on the ROC-AUC results. Considering the time series nature of the EHR database, we tried different approaches to flatten the data. We found that using the data from the last day performed better in terms of ROC-AUC. In contrast, using the mean of all days or the mean of the last three days resulted in worse ROC-AUC.

|  |  |  |
| --- | --- | --- |
|  | Data | ROC-AUC performance |
| Time series flatten | Last day | ✓ |
| All days mean | ✖️ |
| Last three days mean | ✖️ |
| Additional feature | Length of stays | ✓ |
| SD of temporal features | ✖️ |
| Gini impurity of temporal features over all days | ✖️ |

Figure 2. Feature Engineering (get worse: ✖; better: ✓)

Additionally, we tested additional features and observed that incorporating the length of hospital stays improved the ROC-AUC score. However, the standard deviation of temporal features and the Gini impurity of temporal features over all days resulted in decreased score.

We ultimately decided to use the last day record for all 171 features and add 'length of stay' as a new feature. This setting is used for all machine learning algorithms we tried in Section 3.1. For the deep learning part, we further separated 'static features' and 'temporal features', with details included in Section 3.2.

Section 2: Parameter Adjustment

Before delving into the model performance section, this part explains how we identified the best parameters for our top two machine learning algorithms: Gradient Boost and XGBoost (which will be discussed in Section 3 in details). Given the numerous parameters to tune and the slow nature of adjusting each parameter individually, we employed three different search algorithms and k-folder validation. To illustrate the parameter adjustment process, this section uses gradient boosting as an example.

2.1. Searching Algorithms

We employed three different search algorithms for hyperparameter tuning: grid search, random search, and Bayesian optimization. The comparisons of these methods are summarized in Table 3.

|  |  |  |
| --- | --- | --- |
| **Search Method** | **Advantages** | **Disadvantages** |
| **Grid Search** | Systematically searches through all combinations of parameters, ensuring the optimal parameters are found. | Computationally expensive, time-consuming, and can be impractical for many parameters. |
| **Random Search** | Faster than grid search, effectively search in high-dimensional spaces. | Cannot guarantee finding the optimal parameters due to randomness; search results are affected by the random seed |
| **Bayesian Optimization** | Effectively balances exploration and exploitation, suitable for dealing with high-dimensional and non-convex problems | Requires additional computational cost to construct and update the probability model |

Table 3: Comparisons of searching methods

We first tried grid search, which systematically explores the parameter space by evaluating all possible combinations of parameter values. The details of the parameter definitions can be found in Appendix A. However, we found grid search to be very slow due to the exponential growth in the number of combinations.

Next, we tried random search, which explores the parameter space by randomly sampling combinations of parameter values. Although faster than grid search, the random nature of this method does not guarantee finding the optimal parameters, as some combinations might never be sampled.

Finally, building upon the idea of random search, we employed Bayesian optimization. This method uses past evaluation results to form a probability model of the objective function and uses it to select the most promising parameters to evaluate in the next step. This approach effectively balances exploration and exploitation, and it proved to be more efficient and reliable in finding the optimal parameter settings\* in our case.

(\*Our final submitted model version on Kaggle is based on Bayesian optimization results)

2.2. Model Training and Evaluation

To ensure robustness in estimating the model's performance, we applied k-fold cross-validation. Regardless of the searching method used (grid search, random search, or Bayesian optimization), each combination of parameters was used to train the model from k-1 folders’ data, and the performance was evaluated by ROC-AUC score on the kth folder’s data.

The optimal parameter configuration was determined by the parameter values that resulted in the highest average score across all k-folders. Once the optimal parameters were identified, a final model was trained on the entire training dataset to maximize its predictive capabilities.

Section 3: Machine Learning Models

In this section, we summarize our implementations across five machine learning algorithms (Gradient Boost, XGBoost, logistic regression, KNN, and LDA) and five deep learning networks.

3.1 Traditional Machine Learning Algorithm

In this subsection, we present and analyze the performance of various traditional machine learning algorithms used in our study. The results are summarized in Table 4 below:

|  |  |  |
| --- | --- | --- |
| **Model** | **Validation AUC** | **Highlights** |
| Gradient Boosting | 0.7869 | automatic feature selection |
| XGBoost | 0.7742 | automatic feature selection |
| Logistic Regression | 0.7628 | a good trade-off between performance and complexity |
| KNN | 0.7494 | underscore the potential of similarity-based methods |
| LDA | 0.6302 | failed due to its assumption of linear relationship |

Table 4: Results of different models

Our evaluation results indicate that decision tree-based methods, specifically Gradient Boost and XGBoost, yielded the highest performances with AUC scores of 0.7869 and 0.7742, respectively. These methods owe their superior performance to their built-in mechanisms that **automatically identify important features** and capture complex feature interactions. This capacity allows them to effectively handle the intricate relationships within the dataset, leading to improved predictive performance.

Logistic regression, with L1 regularization applied, demonstrated sound performance with a score of 0.7628. Despite its simplicity and speed, it managed to **achieve a good trade-off between performance and complexity,** showcasing its robustness for the task at hand.

K-Nearest Neighbors (KNN) also showed acceptable results with a score of 0.7494. The results from KNN **underscore the potential of similarity-based methods** in this scenario, aligning with the intuitive notion that patients with similar features or hospital records would exhibit similar readmission results.

Linear Discriminant Analysis (LDA), however, exhibited a disappointing performance with a score of 0.6302. This poor performance can be attributed to **its assumption of a linear relationship** within the data. By primarily focusing on linear combinations of features, LDA may not effectively capture the intricate interactions present in the data, ultimately resulting in a subpar performance.

Section 4: Deep Learning Models

In this section, we explore various deep learning algorithms to predict patient readmission rates. With the assumption that neural networks possess robust capabilities for data feature extraction, we conducted minimal feature engineering before deploying these algorithms. The overall results are presented in Table 5, and the details will be discussed in the following subsections.

|  |  |
| --- | --- |
| **Model** | **Validation AUC** |
| LSTM | 0.7665 |
| MLP | 0.7221 |
| GNN | 0.7118 |
| Metric learning | 0.6554 |
| Fusion | 0.6423 |

Table 5: Results of different models

(Note that ‘all features’ in this section refer to 171 features + 'length of stay' incorporated as an additional 172nd feature. Without specification, we use the last day record only.)

4.1. MLP

Our exploration began with the most fundamental type of neural network: the Multilayer Perceptron (MLP).

4.1.1. Method

In our implementation, the MLP adopts a linear-ReLU-linear-sigmoid architecture as shown in Figure 6.

图示

描述已自动生成

Figure 6. Layers of MLP

4.1.2. Observation and Result

After utilizing all the features, the model achieved an ROC-AUC score of 0.7221. However, during training, the loss function did not exhibit a decreasing trend, while the validation AUC only showed a slow increase. This indicates a sign of overfitting, where the model may have learned to perform well on the training data but struggles to generalize to new, unseen data.

4.2. LSTM

To further enhance the performance, we hypothesized that incorporating time series data would provide additional information about the progression of a patient's condition over time. Therefore, we utilized Long Short-Term Memory (LSTM) networks, which are well-suited for handling sequential data that fluctuates over time and may have varying lengths.

4.2.1. Method

We utilized a three-layer Long Short-Term Memory (LSTM) network to predict patient readmission rates. Given the relatively small size of our dataset, we trained our model using two signals:

a) predicting the change in condition score (token change), trained by reconstruction loss (*MSELoss*);

b) predicting the actual prediction (label), trained by prediction loss (*BCEWithLogitsLoss*).

We created a custom LSTM model, which initializes a three-layer LSTM network, with dropout layers included between each layer to prevent overfitting. Additionally, we defined two linear layers—one for predicting the label and another for predicting the token change. At each step, both predictions are generated, and we took the label prediction at the final step as the model's overall prediction.

4.2.2. Observation

Different from the observation of training MLP, the loss function in this case showed a clear decreasing trend, which means our model design is suitable as shown in Figure 7. However, after several attempts, we found that the ROC-AUC score decreased just after the second/third epoch, thus a small epoch number is enough (usually smaller than five).

图表, 折线图

描述已自动生成

Figure 7: Training observations

4.2.3. Result

When utilizing all features, the LSTM model outperformed the MLP, achieving an ROC-AUC score of 0.7456. Moreover, when fed solely with temporal features (specifically, lab test columns), the LSTM model yielded a promising ROC-AUC score of 0.7665. This result confirmed our initial assumption that incorporating temporal features could provide valuable additional information for predicting patient readmission rates.

4.3. Fusion Network

Taking into consideration the nature of our data, we sought to leverage the strengths of each network discussed above. Since LSTM produced promising results with temporal data only, we further proposed a fusion network, integrating the strengths of both the LSTM and MLP model, providing a more comprehensive analysis of both static and temporal features.

4.3.1. Method

This approach considers two groups of features:

Static features (X1): Comprising demographic columns, ICD columns, medication columns, and the length of stays. This amounts to a total of 136 features.

Temporal features (X2): Comprises only the lab test columns, amounting to 36 features.

图示

描述已自动生成

Figure 8. Workflow of Fusion Network

The workflow proceeds as follows (as shown in Figure 8):

1. The static features (X1) undergo processing via an MLP, producing vector v1.
2. The temporal features (X2) are processed through an LSTM, producing vector v2.
3. The resultant vectors (v1 and v2) are then concatenated and fed into an MLP to generate the final output Y.

4.3.2. Observation

Several observations were made during experimentation:

1. Fluctuating Loss: When we used a single Binary Cross-Entropy Loss (*BCELoss*) to update all parameters, we noticed that the loss was always fluctuating. This is not ideal, as a stable or decreasing loss is generally indicative of a model learning effectively.
2. Stagnant AUC: The ROC-AUC score did not increase, suggesting potential overfitting from the first epoch. Overfitting at such an early stage might be indicative of a model that is too complex or a dataset that's too small.
3. Diminished Performance with increased v1/v2 dimensions: Increasing the dimensions of vectors v1/v2 led to significantly poorer performance. In some cases (when using v1=10), the AUC was less than 0.5, which is even worse than random guessing. This indicates the high level features can be represented by simple vectors.

4.3.3. Results

The performance of the fusion network did not meet our expectations, yielding an unsatisfactory ROC-AUC score of 0.6423. This result was much worse than either MLP or LSTM models. One potential reason is that the model might be too complicated. However, even after simplifying the fusion network—replacing the two MLPs with simple linear layers— the score did not exceed 0.7. We suspect that the design of the fusion network may not be able to integrate both side information well.

4.4. Metric Learning

Motivated by the performance with KNN algorithm, we explored Metric Learning, focusing on creating algorithms that learn effective ways to measure similarities or differences between data points. By this approach, we aim to enhance the discernment of data relationships beyond what standard measures offer.

4.4.1. Model Architecture

We employed a Residual Embedding Network for our task. This network was designed to map the input features into an embedding space where the distance between similar points is minimized, and the distance between dissimilar points is maximized.

1. Input and output: The network takes in features of size 172 and maps them to an embedding of dimension v.
2. Layers: The first fully connected layer (fc1) reduces the dimension from 172 to 32. A ReLU activation function follows this layer for non-linearity. The second fully connected layer (fc2) further maps the data to the embedding space of size v.
3. Residual connection: To facilitate training, we incorporated a residual connection. If the input dimension differs from embedding dimension v, an additional linear transformation (shortcut) is used.

4.4.2. Loss Function

We used Triplet Loss, which can guide the model to learn an embedding space where the distance between an anchor and a positive sample (similar) is less than the distance between the anchor and a negative sample (dissimilar) by some margin (we set a margin of 1.0).

4.4.3. Result

However, our results revealed an unexpected outcome. The initial KNN model achieved a promising ROC-AUC of 0.7494, but our metric learning saw a decrease to 0.6554 after one epoch.

This underperformance might be attributed to the limited size of our training dataset and the complexities of hyperparameter tuning in the residual embedding network, indicating a need for further model refinement.

4.5. GNN

After metric learning, we tried another similarity-based method using a Deep Graph Convolutional Network (DeepGCN). This method is particularly effective in learning from data that has inherent graph-like structures.

4.5.1. Graph Construction

Since GNN requires a graph as input, we first used *k neighbors graph* method to connect each data point to its 20 nearest neighbors based on **Euclidean distances**. The number 20 balances detail and computational efficiency. This creates a graph that efficiently captures data relationships for training and inference.

We used masks to preserve the graph structure while preventing data leakage between training and test sets. Masks like *train\_mask, valid\_mask,* and *test\_mask* identify which nodes belong to training, validation, and test datasets, respectively. During training, only nodes with the *train\_mask* are used, with similar approaches for validation and testing. This ensures the model learns from the full graph but accesses only relevant data subsets, maintaining evaluation integrity.

4.5.2. Model Architecture

Our DeepGCN model consists of five Graph Convolutional Network (*GCNConv*) layers. The first four layers transform node features from the input size (*num\_features*) to 32-dimensional hidden layers, ensuring efficient feature processing while keeping the graph's structure. The fifth layer maps these features to the number of classes (*num\_classes*). For regularization, the model uses *ReLU* activation and a dropout rate of 0.5. Additionally, residual connections between layers help counter the vanishing gradient problem in this deep architecture.

*Training Setting:* During the model training, we used the Adam optimizer with a learning rate of 0.01 and trained the model over n epochs using negative log-likelihood loss. For every 100 epochs, we evaluated the model on the validation set.

4.5.3. Observation

When we experimented with increasing the number of adjacent edges beyond 20, we encountered an issue with vanishing gradients due to over-smoothing of the graph convolutions. This suggests a delicate balance in the choice of adjacency in GCNs: while too few edges may not capture enough contextual information, too many can lead to over-smoothing, where node features become too similar, impeding the learning process. These findings highlight the nuanced considerations required in structuring and connecting nodes in GCNs, emphasizing that more complex connectivity doesn't necessarily translate to better performance.

4.5.4. Result

We observed intriguing results after 400 epochs, achieving a ROC-AUC of **0.7118**. However, it still does not outperform KNN. This outcome revealed that higher-order adjacency didn't significantly enhance the model's performance.

Section 5. Discussion

Upon ranking the performance scores from all machine learning and deep learning models (shown in Table 9), it's evident that the tree-based method consistently surpasses all deep learning models. This observation aligns with ongoing debates in literature regarding the comparative efficacy of Gradient Boosted Decision Trees (GBDT) and neural networks.

|  |  |
| --- | --- |
| Model | Validation AUC |
| Gradient Boosting | 0.7869 |
| XGBoost | 0.7742 |
| LSTM | 0.7665 |
| Logistic Regression | 0.7628 |
| KNN | 0.7494 |
| MLP | 0.7221 |
| GNN | 0.7118 |
| Metric learning | 0.6554 |
| Fusion | 0.6423 |
| LDA | 0.6302 |

Table 9. Performance Ranking of All Models

In our study, the best performing deep learning model - LSTM, exhibited similar performance to a basic Logistic Regression (LR) algorithm. **Nonetheless, we maintain our confidence in the potential of neural networks, particularly their prowess in capturing temporal features (as demonstrated by LSTM) and similarity-based information (as seen in metric learning).**

We argue that a more carefully designed fusion network could effectively combine these two types of information, potentially leading to enhanced performance. Moreover, considering the small size of our EHR dataset, we suspect that the neural networks might not have been fully trained, which could also have impacted their performance.

Section 6. Conclusion

Throughout this study, we have utilized 10 distinct traditional machine learning models and deep learning models. Our results indicate that machine learning models demonstrate competitive performance on smaller datasets, while deep learning models are potentially more susceptible to overfitting.

These insights suggest that traditional machine learning models can prove to be reliable choices in scenarios with limited data availability. On the other hand, deep learning models possess unique strengths that traditional machine learning models may not cover, indicating their value in specific situations.

(Word count: 3698)

Appendix

1. **Parameter Definition of Gradient Boost**

The following parameters are considered for adjustment in the Grid Search method:

* 'n\_estimators': This parameter controls the number of weak learners in the ensemble. By decreasing this parameter, the model's complexity is reduced.
* 'learning\_rate': It determines the step size at each boosting iteration. It is advisable to keep this parameter constant or slightly increase it to balance the model's complexity.
* 'max\_depth': This parameter limits the depth of each tree in the ensemble. Reducing this parameter helps prevent overfitting.
* 'min\_samples\_split': It sets the minimum number of samples required to split an internal node. Increasing this parameter ensures that nodes contain more samples, reducing overfitting.
* 'min\_samples\_leaf': This parameter defines the minimum number of samples required to be at a leaf node. Increasing this parameter decreases overfitting.
* 'subsample': It controls the subsampling ratio of the training data for each weak learner. Decreasing this parameter helps prevent overfitting.

1. **Workload Distribution**

The workload is equally divided ☺

1. **Code**

Please refer the coding details in this GitHub link (<https://github.com/lindapu-1/30DaysReadmission/tree/main>)