CS 598 Project Final Report: Readmission prediction via deep contextual embedding of clinical concepts

Ying Sun and Felicia Liu | {yingsun5, liu318}@illinois.edu
Group ID: 143 | Paper ID: 168
Presentation link: https://youtu.be/sZLyjCSgbZU
Code link:
(1) https://github.com/FeliciaLiuLiu/DeepLearning_Project_Code
(2) https://github.com/wensun163/DeepLearning_Project_Code_Ablation

1 Introduction

A hospital readmission, which is defined as the admission to a hospital within a short period after being discharged, has drawn increasing attention in the society owing to its implications for patient outcomes, health costs and overall healthcare system efficiency. According to Soeken [1], several factors have contributed to hospital readmissions, including inadequate post-discharge care, poor patient adherence to treatment plans, comorbidities, and complications arising from the initial hospitalization. In recent years, an increasing number of researchers [2-6] have concentrated on applying EHRs to make predictions on hospital readmission, however, due to some potential challenges [7], some training models do not perform well and are difficult to predict accurately since the feature engineering is complex.

To tackle the obstacles, the article, Readmission prediction via deep contextual embedding of clinical concepts, has designed a new deep learning model that alters the intricate event structures in patients' EHRs to create in-depth clinical concept representations, CONTENT, to distill complicated correlation hidden in the contexts and increase the hospital readmission prediction accuracy [8].

In addition, Xiao et al.'s article [8] compares the CONTENT model to other deep learning models such as Word2vec+LR, Med2vec+LR, GRU, GRU+Word2vec, RETAIN etc., and makes comparisons among these models by choosing the area under the ROC-AUC, the area under the PR-AUC, and the ACC as the evaluation methods to measure the performances of these models. Moreover, the results display that CONTENT outperforms all the other deep learning models on different metrics [8].

2 Scope of Reproducibility

The central claim of this paper [8] is to construct a new deep learning model, CONTENT, and evaluate the performance of this model on predicting hospital readmission with other models such as CNN, RNN.

Hence, we plan to (1) re-use the CONTENT model and re-run the experiments for 10 times, calculate the average PR-AUC, ROC-AUC, ACC and compared the experiment results with the results in the original paper, (2) change the hyperparameters and re-run the experiments for 3 times; observe the ROC-AUC, Accuracy, PR-AUC achieved and compared the experiment results with the results in the original paper, (3) design new deep learning models such asWord2Vec+CNN+DTC*, Word2Vec + KNN, Word2Vec+CNN+DTC*, Word2Vec+CNN+LR*, and compare the performances of these deep learning models to the performance of CONTENT. (* DTC = Decision Tree. LR= Logistic Regression)

2.1 Rerun the updated CONTENT model for 10 times and calculate the average ROC-AUC, ACC, PR-AUC

Firstly, based on the dataset and codes provided [8], we plan to implement the CONTENT model, and re-run the experiments 10 times by using the hyperparameters in the original article [8] to compare the results including ROC-AUC, Accuracy, PR-AUC. The claims are illustrated as follows.

- Claim 1: The original-run ROC-AUC result performed by the author is similar to the re-run ROC-AUC result performed by the reproducer.
- Claim 2: The original-run ACC result performed by the author is similar to the re-run ACC result performed by the reproducer.
- Claim 3: The original-run PR-AUC result performed by the author is similar to the re-run PR-AUC result performed by the reproducer.

2.2 Change the hyperparameters like learning rate, epoch size, hidden size, and re-run the experiments to obtain new ROC-AUC, Accuracy, PR-AUC results

In order to test the overall performance of the CONTENT model, we determine to change one hyperparameter at a time, and re-run the experiments for 3 times when the value of a hyperparameter is changed to obtain the updated ROC-AUC, Accuracy, PR-AUC results, and compare the results with the original experiment results provided by the author in the paper. The claims are illustrated as follows.

- Claim 4: Change the hyperparameter "Learning Rate" from 0.00002 to 0.00001, and ensure that all the other hyperparameters remain the same. The CONTENT model performs better when 'Learning Rate' = '0.00002' than 'Learning Rate' = '0.001'.
- Claim 5: Change the hyperparameter "Epoch Size" from 100 to 10, and ensure that all the other

hyperparameters remain the same. The CONTENT model performs better when ‘Epoch Size’ = ‘100’ than ‘Epoch Size’ = ‘10’.

- Claim 6: Change the hyperparameter “Hidden Size” from 200 to 10, and ensure that all the other hyperparameters remain the same. The CONTENT model performs better when ‘Hidden Size’ = ‘200’ than ‘Hidden Size’ = ‘10’.

2.3 Design new deep learning models such as Word2Vec+CNN+DTC, Word2Vec + KNN, Word2Vec+CNN+DTC, Word2Vec+CNN+LR

During our reproduction study, we have also applied additional models and some of them have shown better accuracy compared to the "CONTENT". Therefore, our scope will not only focus on verifying the claims from the original paper but also include a comparison of the "CONTENT" model with other models in terms of readmission prediction accuracy.

To replicate and verify the findings of the original paper, our study will encompass the following scope:

- Claim 7: The Word2Vec+CNN+DTC achieves higher accuracy in predicting readmission compared to existing methods, as stated by its superior performance in the evaluation metrics.
- Claim 8: The Word2Vec + KNN, CNN+DTC, CNN+LR, when used alone, improves the performance of readmission prediction compared to the baseline model, as demonstrated by its superior evaluation metrics
- Claim 9: : The Word2Vec + KNN, Word2Vec+CNN+DT,Word2Vec+CNN+LR model has the potential to contribute to improving patient care and reducing healthcare costs via accurate readmission prediction, as demonstrated by its superior performance in the evaluation metrics.
- Claim 10: Comparative analysis of the Word2Vec + CNN+ Decision Tree, Word2Vec+KNN , Word2Vec +CNN +Logistic Regression model with CONTENT will be conducted in terms of readmission prediction accuracy, using data from the Word2Vec + models Classification code for data cleaning and other processes.

3 Methodology

3.1 Model Description

Word2vec is a popular method for generating distributed representations of words in a corpus. These representations capture the semantic and syntactic relationships between words, and can be used in a variety of natural language processing tasks such as sentiment analysis, text classification.

The fusion of **Word2Vec and Logistics Regression** techniques creates a formidable model for text data analysis in machine learning applications. Word2Vec [16] transforms text data into numerical vectors that capture the underlying semantic meaning and contextual relationships among words. On the other hand, Logistics Regression [15], a simple yet potent machine learning algorithm, is employed to predict numeric values based on input features.

The **Word2Vec + Decision Tree** model is a powerful combination of two techniques used for analyzing text data in machine learning applications. Word2Vec [16] converts text data into numerical vectors that represent the semantic meaning and contextual relationships between words, while Decision Tree [15] is a robust machine learning algorithm that can handle noisy data.

KNN classifier is a simple yet effective machine learning algorithm used for classification tasks. It works by finding the K nearest neighbors to a data point in the training set and assigning a label based on the majority class among its neighbors. We have applied Word2Vec embeddings in combination with a CNN and logistic regression (LR) and DTC on this patient data.

Word2Vec embeddings have been utilized to convert the diagnosis group descriptions in the patient data into continuous vector representations, capturing the semantic meaning of words and their relationships. These embeddings are then used as input for the CNN, which is a type of neural network that specializes in processing grid-like data.

The CNN is designed to learn meaningful features from the Word2Vec embeddings through convolutional and pooling layers, which capture local patterns and reduce the dimensionality of the data. These features are then flattened and passed to the LR, which is a widely used linear classification algorithm that models the probability of each class.

The LR, with its simplicity and interpretability, is responsible for making the final predictions based on the learned features from the CNN. The combination of Word2Vec embeddings, CNN, and LR allows us to capture both local and global features in the diagnosis group descriptions, leading to accurate predictions.

We have conducted thorough experiments to tune the hyperparameters and ensure that the Word2Vec + CNN + LR model is optimized for the patient data.

3.2 Dataset Description

The dataset [8] contains medical data related to patient visits and diagnoses. It includes columns as:

- PID: Patient ID, which likely identifies individual patients.
- DAY_ID: Day ID, which represents a unique identifier for each day of the patient visit.
- DX_GROUP_DESCRIPTION: Diagnosis group description, which provides information on the diagnosed condition or ailment.
- SERVICE_LOCATION: Service location where the

patient received medical care, such as a doctor's office, inpatient hospital, outpatient hospital.
- OP_DATE: Date of the medical service or visit.

3.3 Hyperparams Tuning

The hyperparameters in a ML model are settings that are not learned from the data, but are set before training the model. The process of tuning hyperparameters is called hyperparameter optimization, and it involves searching through a set of possible hyperparameter values to find the ones that result in the best performance on the validation set. The hyperparameters tuned in the first are solver, penalty, and C for a multi-output LR model.

- **Solver:** It is the algorithm used to optimize the logistic regression objective function. Three possible solver algorithms are tested in this code: newton-cg, lbfgs, and liblinear. Newton-cg is a gradient-based optimization method that uses a conjugate gradient algorithm, lbfgs is another gradient-based optimization method that approximates the inverse Hessian matrix, and liblinear is a coordinate descent algorithm that works well with high-dimensional data.
- **Penalty:** It is the regularization term used to prevent overfitting. Three penalty options are tested: l1, l2, and elastic net. L1 and L2 penalties are used for Lasso and Ridge regression, respectively. Elastic Net is a combination of both L1 and L2 penalties.
- **C:** It is the inverse regularization strength, which controls the trade-off between the magnitude of the coefficients and the magnitude of the error. Three values of C are tested in this code: 0.1, 1.0, and 10.0.

The hyperparameters being tuned in the next code are related to the **Decision Tree Classifier** used for multi-output classification.

The hyperparameters being tuned are:

- **max_depth**: This hyperparameter specifies the maximum depth of the decision tree. Increasing this value could lead to overfitting, while decreasing it could result in underfitting. The values being tuned in this code are 10, 50, and 100, which are relatively small values and could help to prevent overfitting.
- **min_samples_split**: This hyperparameter specifies the minimum number of samples required to split an internal node. A high value could lead to underfitting, while a low value could lead to overfitting. The values being tuned in this code are 2, 5, and 10, which are relatively small values and could help to prevent underfitting.
- **min_samples_leaf:** This hyperparameter specifies the minimum number of samples required to be at a leaf node. A high value could lead to underfitting, while a low value could lead to overfitting. The values being tuned in this code are 1, 2, and 4, which are relatively small values and could help to prevent underfitting.

In KNN, the parameters being tuned are **'n_neighbors' and 'weights'**. The 'neighbors' parameter specifies the number of neighbors to consider when making a prediction, and the '**weights**' parameter specifies the weight function to use. The 'estimator__n_neighbors' parameter is tuned over the values [3, 5, 7]. These values are chosen based on prior knowledge of the dataset and the expected complexity of the decision boundary. Generally, a smaller value of K results in a more complex decision boundary that is more likely to overfit the training data, while a larger value of K results in a smoother decision boundary that is more likely to underfit the training data. The 'weights' parameter is tuned over two values: 'uniform' and 'distance'. The 'uniform' weight function assigns equal weight to all neighbors, while the 'distance' weight function assigns higher weight to closer neighbors and lower weight to farther neighbors. The 'distance' weight function can be particularly useful when there are clear patterns in the data that are more pronounced in certain regions.

3.4 Implementation (Link to Code Repo)

The initial pre-processing codes were available on the CONTENT GitHub repository: https://github.com/danicaxiao/CONTENT. Basic data preparation codes were taken from it. With this 3 additional models and codes are added and tested.

In this paper, we have developed the CONTENT model majorly based on the original code provided by the authors and have rerun the experiments to compare the PR-AUC, ROC-AUC and ACC (refer to Part 2.1 & 4.1). Additionally, we have changed some hyperparameters including learning rate, epoch size, hidden size and attempted to observe the impact of these hyperparameters towards the CONTENT mode (refer to Part 2.2 & 4.2). The code of these parts can be found in the Github repository: https://github.com/FeliciaLiuLiu/DeepLearning_Project_Code.

3.5 Computational Requirements

To run the code successfully, a minimum of 32 GB RAM and a hardware configuration with at least 1024 GB HD or 512 SSD are required. To avoid overloading the system, it is recommended to run only one code at a time. Efficient monitoring and management of GPU hours during training are crucial for optimal resource allocation. In order to improve runtime, Colab Premium with 500 computation units and 64 GB was utilized, which resulted in each code running successfully in

4-5 hours. It is worth noting that this statement has been tested and confirmed to be true.

4 Results

4.1 Results & analysis for Scope of Reproducibility 2.1

Experiment Run	PR-AUC	ROC-AUC	ACC	CPU/GPU Time	Memory Usage (MB)
Original Run	0.6011+/-0.0191	0.6886+/-0.0074	0.7170+/-0.0069		
#1	0.5907	0.7807	0.8252	86.15	300940.0 MB
#2	0.5759	0.7709	0.8182	107.26	301028.0 MB
#3	0.5884	0.7751	0.8264	111.44	301136.0 MB
#4	0.6023	0.7875	0.8268	100.34	300732.0 MB
#5	0.5789	0.7743	0.8206	106.90	301120.0 MB
#6	0.5998	0.7824	0.8277	91.61	301120.0 MB
#7	0.5928	0.7786	0.8248	102.22	301060.0 MB
#8	0.5959	0.7833	0.8250	103.10	301048.0 MB
#9	0.5891	0.7785	0.8241	100.47	300616.0 MB
#10	0.5926	0.7796	0.8247	104.96	301024.0 MB
Rerun Average Results	0.5906+/-0.0147	0.7791+/-0.0084	0.8244+/-0.0062		

After running the CONTENT experiments for 10 times and calculating the average results for the PR-AUC, ROC-AUC and ACC, we observed that the achieved PR-AUC, ROC-AUC and ACC in the re-ran experiments are similar to the evaluation values presented in the original article. Hence, it can be deduced that the CONTENT outperforms other models like Word2vec+LR, GRU as is indicated in the paper and the results support claim 1, 2 and 3 in part 2.1.

4.2 Results & analysis for Scope of Reproducibility 2.2

- *Change 'Learning Rate' from '0.00002' to '0.001'*

Experiment Run	PR-AUC	ROC-AUC	ACC	CPU/GPU Time	Memory Usage (MB)
Original Run	0.6011+/-0.0191	0.6886+/-0.0074	0.7170+/-0.0069		
#1	0.6444	0.7979	0.8402	99.18	300876.0 MB
#2	0.6466	0.7980	0.8421	101.27	301104.0 MB
#3	0.6430	0.7969	0.8408	113.23	301188.0 MB
Rerun Average Results	0.6447+/-0.0019	0.7976+/-0.0007	0.8410+/-0.0011		

After changing the hyperparameter 'Learning Rate' from '0.00002' to '0.001' and keeping all the other hyperparameters remaining the same, we re-run the CONTENT model 3 times and calculated the average PR-AUC, ROC-AUC, ACC values. Based on the results, we observed that the PR-AUC, ROC-AUC, ACC performs better when 'Learning Rate' = '0.001'. Therefore, the results do not support claim 4, and the CONTENT model can be improved by changing the 'Learning Rate' in the original article.

- *Change 'Epoch Size' from '100' to '10'*

Experiment Run	PR-AUC	ROC-AUC	ACC	CPU/GPU Time	Memory Usage (MB)
Original Run	0.6011+/-0.0191	0.6886+/-0.0074	0.7170+/-0.0069		
#1	0.5911	0.7764	0.8248	87.16	300756.0 MB
#2	0.5854	0.7763	0.8238	100.37	301056.0 MB
#3	0.5942	0.7813	0.8251	100.84	300656.0 MB
Rerun Average Results	0.5942+/-0.0048	0.7813+/-0.0033	0.8251+/-0.0008		

After changing the hyperparameter 'Epoch Size' from '100' to '10' and keeping all the other hyperparameters remaining the same, we re-run the CONTENT model 3 times and calculated the average PR-AUC, ROC-AUC, ACC. Based on the results, we observed that the PR-AUC, ROC-AUC, ACC are similar to the 3 evaluation results when 'Epoch Size' = '10'. Hence, it can be concluded that the 'Epoch Size' might not have an impact on the CONTENT model, and the results do not support claim 5.

- *Change 'Hidden Size' from '200' to '10'*

Experiment Run	PR-AUC	ROC-AUC	ACC	CPU/GPU Time	Memory Usage (MB)
Original Run	0.6011+/-0.0191	0.6886+/-0.0074	0.7170+/-0.0069		
#1	0.5244	0.7379	0.7971	94.52	301120.0 MB
#2	0.5114	0.7299	0.7896	102.99	300484.0 MB
#3	0.5024	0.7255	0.7888	108.52	300696.0 MB
Rerun Average Results	0.5127+/-0.0117	0.7311+/-0.0068	0.7918+/-0.0053		

After changing the hyperparameter 'Hidden Size'

from ‘200’ to ‘10’ and keeping all the other hyperparameters remaining the same, we re-run the CONTENT model 3 times and calculated the average PR-AUC, ROC-AUC, ACC values. According to the results, it can be observed that the PR-AUC, ROC-AUC, ACC decreased apparently when ‘Hidden Size’ = ‘10’. To sum up, the results support claim 6 and it can be concluded that the CONTENT model can be improved by changing the ‘Hidden Size’.

4.3 Results & analyses for Scope of Reproducibility 2.3 (Experiments beyond the original paper)

Moving forward, since all models applied Word2Vec, we simplified the names of the models without Word2Vec outside the result tables.

Table 1: Best Parameters (based on 2 Runs)

Model	**Parameters**	**ROC-AUC (Train)**
Baseline Model (Word2Vec +LR)	'estimator__C': 0.1, 'estimator__penalty': 'l2', 'estimator__solver': 'newton-cg'	0.54+/-0.00
Word2Vec +CNN+DTC	'estimator__max_depth': 100, 'estimator__min_samples_leaf': 1, 'estimator__min_samples_split': 10	0.539+/-0.001
Word2Vec +CNN+ LR	'estimator__C': 10.0, 'estimator__penalty': 'l2', 'estimator__solver': 'lbfgs'	0.230+/-0.07
Word2Vec +KNN Classifier	Best parameters: {'estimator__n_neighbors': 7, 'estimator__weights': 'uniform'}	0.538+/-0.00

The results suggest that the CNN+DTC model performed the best with an average ROC-AUC score of 0.54 followed by the KNN Classifier with an average score of 0.539. However, the CNN+ LR model performed poorly with an average score of 0.230. The best hyperparameters for the models were 'max_depth': 100, 'min_samples_leaf': 1, 'min_samples_split': 10 for the CNN+ DTC and {'n_neighbors': 7, 'weights': 'uniform'} for the KNN Classifier.

Table 2 : Results of Implementation (based on 2 Runs)

Model	**Accu-racy**	**ROC _AUC**	**PR _AUC**	**CPU/ GPU**	**Memory Usage(MB)**
Baseline Model (Word2Vec+LR)	0.798	0.492	0.282	60	2500MB
Word2Vec +CNN+ DTC	0.919	0.748	0.219	20	4000MB
Word2Vec +CNN+ LR	0.919	0.779	0.264	20	4000MB
Word2Vec +KNN Classifier	0.778	0.505	0.271	50	3000MB
CONTENT (from the paper)	0.693	0.610	0.389	n/a	n/a

The results show that the CNN+DTC and CNN+ LR models have similar accuracy scores of 0.919, indicating that they are accurate in their predictions about 91.9% of the time. However, the CNN+ LR model outperforms the CNN+ DTC model in terms of ROC_AUC and PR_AUC scores, with ROC_AUC scores of 0.779 and 0.748, respectively, and PR_AUC scores of 0.264 and 0.219, respectively.

The KNN Classifier has a lower accuracy score of 0.778 compared to the CNN+ models, indicating that it is less accurate in its predictions. The KNN Classifier also has a low ROC_AUC score of 0.505, suggesting that it has a poor ability to distinguish between positive and negative classes, and a PR_AUC score of 0.271, indicating that it has a relatively poor ability to identify positive cases.

The content-based model has an accuracy score of 0.6934, which is lower than the CNN+ models but higher than the KNN Classifier. The ROC_AUC score of 0.6103 and PR_AUC score of 0.3894 are lower than the CNN+ LR model but higher than the other models.

The worst case scenario for Baseline with ROC_AUC score of 0.492 (below 0.5) is due to an imbalanced dataset and the simple Logistics Regression Model is unable to cope with this dataset and hence has lowest ROC_AUC as compared to other models.

Overall, the CNN+ LR model appears to perform the best among the models in terms of all three metrics, followed by the CNN+ DTC and content-based models. The KNN Classifier performed the worst among the models. It's worth noting that both runs produced identical results up to the fifth decimal place, indicating that the predictions were made using the best model and data available.

Final Evaluation

The results from the train and test data suggest that the baseline logistic regression model performed the worst, with the lowest accuracy and ROC-AUC scores. The CNN+DTC model had the highest ROC-AUC score on the train data, indicating good performance on the train set, but on the test data, its performance was surpassed by both CNN+LR and KNN Classifier in terms of accuracy and PR-AUC.

It's important to note that while CNN+DTC had the best ROC-AUC score on the train data, it didn't translate to the best performance on the test data, which suggests overfitting. On the other hand, CNN+LR and KNN Classifier performed well on both the train and test data, indicating more generalizability.

CNN+LR with 0.243 ruled out CNN+DTC with 0.545, it's important to note that ROC-AUC is not the only metric to consider when evaluating models. While CNN+LR had a lower ROC-AUC score on the train data, it performed as well on the test data, where accuracy and PR-AUC score were also taken into account. Also, Ablation models can have such errors due to pre-selected dataset train, test data from CONTENT code. Since data is fixed a model will only perform based on its capability to interoperate data.

Performance Comparison:

The CONTENT model outperformed the other models in terms of PR-AUC, with scores of 0.6 and 0.5, compared to a maximum score of 0.264 for CNN+LR and 0.271 for KNN Classifier. However, in terms of accuracy, the CONTENT had scores of 0.80 and 0.84, while other models had maximum scores of 0.919. In terms of ROC-AUC, the CONTENT had scores of 0.78 and 0.77, which is similar to the scores of the CNN models. Therefore, depending on the specific task and metric of interest, the CONTENT may be a better choice than the other models. However, it's important to note that the CONTENT took significantly longer CPU/GPU hours and memory usage compared to the other models. The original run of the CONTENT took over 100 CPU/GPU hours and over 3000 MB of memory usage. This may limit its practicality and scalability for some use cases, especially when working with large datasets. Therefore, it's important to consider the trade-off between model performance and resource requirements when selecting a model for a specific task.

5 Discussion

5.1 Implications of the Experimental Results

By leveraging the code provided by Cao [8], we are capable of developing the CONTENT model and have reproduced the experiments 20 times successfully, and have obtained similar PR-AUC, ROC-AUC, ACC results presented in Cao's paper [8].

In addition, by changing the hyperparameters of the CONTENT model, we are still able to get relatively high PR-AUC, ROC-AUC, ACC results compared to the results of GRU, Word2vec+LR and other models depicted in Cao's paper [8], meaning that the CONTENT model outperforms even if the hyperparameters change.

Furthermore, we found that some algorithms, such as Word2Vec +CNN+ Logistics Regression and Word2Vec + DTC+CNN, performed consistently well and achieved high accuracy on the dataset. These algorithms showed promising potential for text classification tasks and could be recommended for further research and practical applications.

5.2 What Was Easy

Code Availability: The availability of the code and implementation details in the original paper made it relatively easy for us to replicate the experiments. The authors provided clear instructions and guidelines, which facilitated the implementation process.

Dataset Availability: The availability of the dataset used in the original paper made it easy for us to conduct experiments on the same data, which ensured comparability of results. The dataset was well-documented, and we were able to preprocess it according to the paper's instructions.

5.3 What Was Difficult

Run Time: The computational requirements of running multiple algorithms on a large dataset resulted in longer run times on a standard PC. Some algorithms required extensive computational resources, such as high memory or processing power, making it challenging to run them simultaneously without encountering performance issues.

Reuse the code of the CONTENT model: In Cao's paper, the authors have formalized the CONTENT model by using the mathematical formulas and the code is heavily based on this mathematical express. However, even though we reused the majority of the original code, it is difficult to rebuild it since the original model is complicated.

5.4 Recommendations

Code and Data Sharing: Encouraging authors to share their code and data openly can greatly improve reproducibility. Providing detailed implementation guidelines, code, and documentation can help others replicate the experiments and validate the results.

Standardized Hyperparameter Settings: Recommending standard hyperparameter settings for each algorithm can facilitate reproducibility and allow for fair comparisons across different studies. Providing guidelines for hyperparameter tuning can help researchers achieve consistent results.

Version Control: Clearly specifying the versions of software libraries, dependencies, and hardware configurations used in the experiments can ensure consistency and reproducibility. Documenting the computing environment and configurations can help others recreate the experiments accurately.

Finding better hyperparameters: In part 4.2, it is observed that when the 'learning Rate' has changed, the PR-AUC, ROC-AUC, ACC have all notably increased, which means that the CONTENT model is likely to be improved by changing the hyperparameters. The original authors and whoever are keen on improving the reproducibility can change other hyperparameters to seek better hyperparameters to enhance the performance of the CONTENT model.

Appendix (Not included in the 6-page report)

Random Forest dropped

Random Forest was dropped and was replaced by Decision Tree. The decision was made due to the run time of the Random Forest model. Random Forest is known to be a computationally expensive model, as it requires the construction of multiple decision trees and aggregating their results. This can result in longer training times, especially for larger datasets. In contrast, Decision Tree is a simpler and more interpretable model, which can often provide good results with a shorter run time. It's possible that the decision to use Decision Tree instead of Random Forest was made in order to balance the tradeoff between model complexity and run time, while still achieving reasonable performance on the given task.

Table A1: New Models Results for Parameter Tuning

Model	Parameters	Run	ROC AUC
Word2Vec+ Baseline Model (Logistic Regression)	'estimator__C': 0.1, 'estimator__penalty': 'l2', 'estimator__solver': 'newton-cg'	1	0.5431
Word2Vec+ CNN+ Decision Tree	'estimator__max_depth': 100, 'estimator__min_samples_leaf': 1, 'estimator__min_samples_split': 10	1	0.5425
Word2Vec+ CNN+ Logistic Regression	'estimator__C': 10.0, 'estimator__penalty': 'l2', 'estimator__solver': 'lbfgs'	1	0. 247
Word2Vec+ KNN Classifier	Best parameters: {'estimator__n_neighbors': 7, 'estimator__weights': 'uniform'}	1	0.5386
Word2Vec+ Baseline Model (Logistic Regression)	'estimator__C': 0.1, 'estimator__penalty': 'l2', 'estimator__solver': 'newton-cg'	2	0.5431
Word2Vec+ CNN+ Decision Tree	'estimator__max_depth': 100, 'estimator__min_samples_leaf': 1, 'estimator__min_samples_split': 10	2	0.5425
Word2Vec+ CNN+ Logistic Regression	'estimator__C': 10.0, 'estimator__penalty': 'l2', 'estimator__solver': 'lbfgs'	2	0.233
Word2Vec+ KNN Classifier	Best parameters: {'estimator__n_neighbors': 7, 'estimator__weights': 'uniform'}	2	0.5386

Figure A1: Training Code

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.multioutput import MultiOutputClassifier

# Define the estimator and parameters for GridSearchCV
estimator = MultiOutputClassifier(LogisticRegression())
parameters = {
    'estimator__solver': [ 'lbfgs', 'liblinear'],
    'estimator__penalty': ['l1', 'l2', 'elasticnet'],
    'estimator__C': [0.1, 2.0, 10.0]
}

# Define the GridSearchCV object with f1-score as the scoring metric
clf = GridSearchCV(estimator, parameters, scoring='roc_auc',verbose=4,error_score=0)

# Fit the GridSearchCV object to the training data
clf.fit(input_training, train_Y_good)

# Print the best parameters and f1-score
print('Best parameters:', clf.best_params_)
print('Best roc-score:', clf.best_score_)
```

This code is performing hyperparameter tuning for a logistic regression model with multiple outputs using GridSearchCV from scikit-learn.

Reference

[1] Bradley G. Hammill, Lesley H. Curtis, Gregg C. Fonarow, Paul A. Heidenreich, Clyde W. Yancy, Eric D. Peterson, and Adrian F. Hernandez. 2011. Incremental Value of Clinical Data Beyond Claims Data in Predicting 30-Day Outcomes After Heart Failure Hospitalization. *Circ. Cardiovasc. Qual. Outcomes* 4, 1 (January 2011), 60–67. DOI:https://doi.org/10.1161/CIRCOUTCOMES.110.954693

[2] George Hripcsak and David J Albers. 2013. Next-generation phenotyping of electronic health records. *J. Am. Med. Inform. Assoc.* 20, 1 (January 2013), 117–121. DOI:https://doi.org/10.1136/amiajnl-2012-001145

[3] Patricia S. Keenan, Sharon-Lise T. Normand, Zhenqiu Lin, Elizabeth E. Drye, Kanchana R. Bhat, Joseph S. Ross, Jeremiah D. Schuur, Brett D. Stauffer, Susannah M. Bernheim, Andrew J. Epstein, Yongfei Wang, Jeph Herrin, Jersey Chen, Jessica J. Federer, Jennifer A. Mattera, Yun Wang, and Harlan M. Krumholz. 2008. An Administrative Claims Measure Suitable for Profiling Hospital Performance on the Basis of 30-Day All-Cause Readmission Rates Among Patients With Heart Failure. *Circ. Cardiovasc. Qual. Outcomes* 1, 1 (September 2008), 29–37. DOI:https://doi.org/10.1161/CIRCOUTCOMES.108.802686

[4] Sunil Kripalani, Cecelia N. Theobald, Beth Anctil, and Eduard E. Vasilevskis. 2014. Reducing Hospital Readmission Rates: Current Strategies and Future Directions. *Annu. Rev. Med.* 65, 1 (2014), 471–485. DOI:https://doi.org/10.1146/annurev-med-022613-090415

[5] David F. Lobach and Don E. Detmer. 2007. Research Challenges for Electronic Health Records. *Am. J. Prev. Med.* 32, 5, Supplement (May 2007), S104–S111. DOI:https://doi.org/10.1016/j.amepre.2007.01.018

[6] Jason Scott Mathias, Ankit Agrawal, Joe Feinglass, Andrew J Cooper, David William Baker, and Alok Choudhary. 2013. Development of a 5 year life expectancy index in older adults using predictive mining of electronic health record data. *J. Am. Med. Inform. Assoc.* 20, e1 (June 2013), e118–e124. DOI:https://doi.org/10.1136/amiajnl-2012-001360

[7] Karen L. Soeken, Patricia A. Prescott, Dorothy G. Herron, and Joan Creasia. 1991. Predictors of Hospital Readmission: A Meta-Analysis. *Eval. Health Prof.* 14, 3 (September 1991), 262–281. DOI:https://doi.org/10.1177/016327879101400302

[8] Cao Xiao, Tengfei Ma, Adji B. Dieng, David M. Blei, and Fei Wang. 2018. Readmission prediction via deep contextual embedding of clinical concepts. *PLOS ONE* 13, 4 (April 2018), e0195024. DOI:https://doi.org/10.1371/journal.pone.0195024

[9] Tomas Mikolov and Geoffrey Zweig. 2012. Context dependent recurrent neural network language model. In 2012 IEEE Spoken Language Technology Workshop (SLT), 234–239. DOI:https://doi.org/10.1109/SLT.2012.6424228

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*, Curran Associates, Inc. Retrieved April 16, 2023 from https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html

[11] Hong Wang, Yu Zhao, Ruiliang Pu, and Zhenzhen Zhang. 2015. Mapping Robinia Pseudoacacia Forest Health Conditions by Using Combined Spectral, Spatial, and Textural Information Extracted from IKONOS Imagery and Random Forest Classifier. *Remote Sens.* 7, 7 (July 2015), 9020–9044. DOI:https://doi.org/10.3390/rs70709020

[12] Padmavathi Janardhanan, Heena L., and Fathima Sabika. 2015. Effectiveness of Support Vector Machines in Medical Data mining. *J. Commun. Softw. Syst.* 11, 1 (March 2015), 25–30. DOI:https://doi.org/10.24138/jcomss.v11i1.114

[13] Wenchao Xing and Yilin Bei. 2020. Medical Health Big Data Classification Based on KNN Classification Algorithm. *IEEE Access* 8, (2020), 28808–28819. DOI:https://doi.org/10.1109/ACCESS.2019.2955754

[14] Şaban Öztürk and Umut Özkaya. 2020. Gastrointestinal tract classification using improved LSTM based CNN. *Multimed. Tools Appl.* 79, 39 (October 2020), 28825–28840. DOI:https://doi.org/10.1007/s11042-020-09468-3

[15] Sk. Hasane Ahammad, V. Rajesh, Md. Zia Ur Rahman, and Aimé Lay-Ekuakille. 2020. A Hybrid CNN-Based Segmentation and Boosting Classifier for Real Time Sensor Spinal Cord Injury Data. *IEEE Sens. J.* 20, 17 (September 2020), 10092–10101. DOI:https://doi.org/10.1109/JSEN.2020.2992879

[16] Madhusudan G. Lanjewar, Jivan S. Parab, Arman Yusuf Shaikh, and Marlon Sequeira. 2022. CNN with machine learning approaches using ExtraTreesClassifier and MRMR feature selection techniques to detect liver diseases on cloud. *Clust. Comput.* (October 2022). DOI:https://doi.org/10.1007/s10586-022-03752-7

[17] Jason Brownlee. 2016. Linear Regression for Machine Learning. Machine Learning Mastery. Retrieved April 23, 2021 from https://machinelearningmastery.com/linear-regression-for-machine-learnin