Impact of Hospital Health Record on the Remission of Diabetes Inpatients

Yu Xiang Zhang Bo Zhang

Abstract

We present a classification problem for a clinical database with 101767 encounter records with 50 columns each. The idea is to predict whether a Diabetes patient will be readmitted. The problem is simplified to binary classification after observing the imbalance among the 3 classes. There are 49 features with some being irrelevant to the goal, so we drop the features that will most likely not contribute to the prediction. The relevant features could be human characteristics, medication treatment, or another test result. After data wrangling, the preprocessed data were trained with four different classifiers: Decision Tree, Random Forest, SVM, Neural Network. Comparing the performance of the algorithms, results show that Random Forest gave the best accuracy, which is 0.64, and cost significantly less training time than SVM and Neural Network. However, there is no model that could achieve our goal which is above 90% accuracy.

1. Introduction

Given a sample of patient encounter, we extract information such as race, age, gender, number of inpatient visits, number of emergency visits, A1c test result, medications taken, etc. We convert this information into numerical data to feed into the classification models. The predicted result of the models will be either NO for no readmission or YES for readmission in less or more than 30 days. The preliminary analysis heavily relies on the general information presented in the [7]. The methodology of data wrangling comes from what was presented in the course of Data Analytics. The hyperparameter searching technique is learned in the Machine Learning course, with reference to online documentation primarily at [1].

2. Methodology & Experimental Results

2.1. Data Wrangling

The first thing to worry about should be the quality of data. The Pandas Python library helps analyze the following aspects of the diabetics dataset.

First of all, the collected dataset must provide value to the objective. Which of the fifty features and how do they add useful information to predicting the readmission of the patients? Looking through the feature names and their description in Table 1 of [7] provide a general sense as to how they contribute to the prediction without delving into the expert knowledge of physicians. Features can be separated into the following types: unique identifiers, coded identifiers, physiological characteristics, incident/event count by the hospitals, medications taken by the patients. Unique identifiers can generally be disregarded because they are randomly assigned, except that the "patient_nbr" is used to identify patients which may have many "encounter_id"s registered on file. According to [4], the "patient_nbr" can be used to group these samples when splitting to make sure that each group only appears once in either training or testing dataset. To achieve such splitting strategy, we used GroupShuffleSplit from scikit-learn library. Coded identifiers need proper encoding; generally the best way is to encode them using One-Hot Encoding (binarize every unique integer value). However, considering that there are already over 40 features, we decide not to add new features to our dataset and simply encode ordinally using Pandas category codes. The rest of the features are part of the domain expertise, and thus we want to keep them as much as possible.

Second, each of the feature data must be inside valid ranges

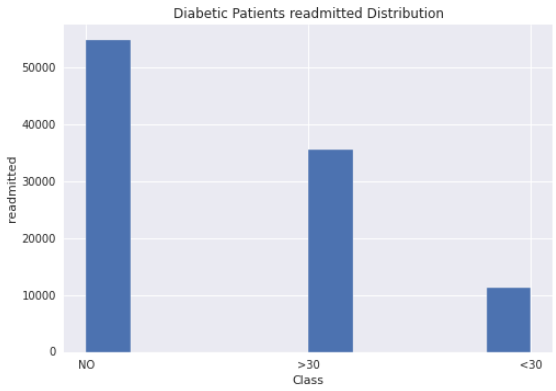


Figure 1: Class Distribution: 3 Initial Imbalanced Classes.

and be of valid data types. Does the data type and the value range look reasonable and natural for a particular feature? The data types are also expected to be uniform. What measures should be taken to encode the categorical data?
We first separate the features based on their dtype — there are only 2. The "object" dtype generally indicates a category, except for "age" and "weight" which are naturally numerical. The "int64" dtype generally indicates a numerical feature, except for "admission_type_id", "discharge_disposition_id", and "admission_source_id". The latter are annotated in the id_mapping.csv file and should be treated as categorical. The features "medical_specialty", "diag_1", "diag_2", "diag_3" contain over 50 unique values; because this makes them harder to encode into numerical data, we decide to drop them. The features "examide", "citoglipton" have one unique value and are hence not useful to prediction.
Third, the dataset must be complete because the machine learning models cannot handle missing values during training. How to deal with a feature with a high percentage of missing values? A preliminary inspection indicated the presence of many categorical features and some with high percentage missing values. These missing values are identified by the character "?" in the original dataset and are replaced by np.nan when imported as a Pandas DataFrame. The features with less than 5 percent missing values were kept and imputed with the most frequent values, which is achieved by scikit-learn SimpleImputer. Those with more missing values were dropped. In hindsight, flagging would preserve the information and is considered to be a better approach in general. When flagging, all the missing values may be grouped under a new value "MISSING", or the category that contains these missing values may be converted to a new category that indicates whether the corresponding value is missing or not.

Garbage in, garbage out. We were aware of this principle before starting this project. However, it is not after we trained a base MLP model that we realized the importance of data cleaning. For this first base model, we ignored the presence of irrelevant features and included them in the training data. The categorical features were encoded ordinally as how the values appeared in the dataset. This first model yielded an accuracy of about 54% for 3 classes (NO, ¡30, ¿30) as shown in Figure 1; it basically predicted "NO" for every input. In reaction to this poor result, we revised the data wrangling step and cleaned data further using the techniques presented above.

2.2. Preprocessing

The input data are expected by many models to be in the same value range. In addition to the SimpleImputer mentioned above, the preprocessing also involves a scikit-learn StandardScaler that scales the data to 0 mean and unit variance. It is important to only use training data when computing the mean and variance, i.e. fitting the Scaler only on training data. Otherwise, the testing data may no longer indicate true performance on unseen data.

2.3. Hyperparameter Tuning

As advertised in the proposal, the dataset is trained on Decision Tree, Neural Network and Support Vector Machine models. The training process inevitably begs the question of hyperparameter optimization.
Based on [2] and due to time constraint, we abandon the idea of Grid Search and proceed directly with Randomized Search. In retrospection, Grid Search could probably provide a list of reliable hyperparameter candidates before diving into Randomized Search. During our experiment with Randomized Search, we found that tuning some parameters does not yield a good performance, i.e. they should be left unchanged. This explains why most of our tuned models cannot even out-perform the base model. Generally speaking, we fail at assessing the importance of the hyperparameters.
As the Randomized Search time becomes longer for the MLP (neural network) and the SVM model, we took several measures to speed the parameter searching up. First, we can reduce the search space by cutting the number of hyperparameters and the number of iterations. Second, based on [6], we can use Bayesian optimization techniques which take experimental results into account. This idea is realized in the library [5] and backed up by [3]. We choose this library because we are not really familiar with the idea of Bayesian optimization, but HyperOpt provides an easy API to automate the searching process and allow us to track its progress via its output. Lastly, because the dataset contains tens of thousands of samples, it proves to be useful to limit the training dataset size to a tenth of the original dataset (around 10,000 samples) and still get roughly the same performance on the testing set.

2.4. Metrics and Evaluation

With hyperparameter tuning we often use cross-validation to estimate the performance. During Randomized Search, we adopt the GroupKFold strategy which is similar to K-fold but based on the groups of "patient_nbr" mentioned above. When we introduce HyperOpt-sklearn in our parameter search, due to the lack of documentation, we abandon the group-based CV strategy but instead use the default K-fold. As a result of this discontinuity, we fail to identify the difference in performance between the two CV strategies.

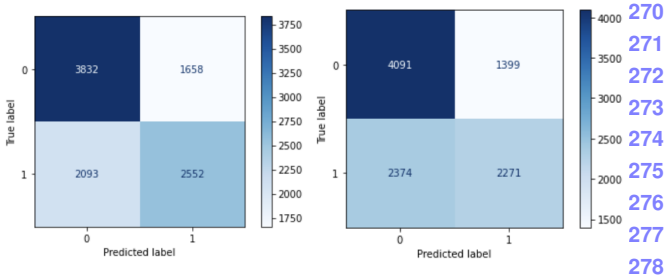
	precision	recall	f1-score	support
0	0.60	0.83	0.70	5490
1	0.64	0.36	0.46	4645
accuracy			0.61	10135
macro avg	0.62	0.59	0.58	10135
weighted avg	0.62	0.61	0.59	10135

Table 1: Classification Report for Best Random Forest Model.

We began our classification with the original three classes (NO, >30, <30) representing no readmission, readmission beyond 30 days and within 30 days. However, the preliminary model would never predict one of the three classes (>30). In reaction to this result, we decided to simplify the problem by converting the three classes into two, effectively merging the >30 and <30 by YES. The target labels then become NO or YES whether the patient is subject to readmission. In terms of loss functions, we used the ROC AUC, F1 with macro averaging and weighted averaging. Since we turn the problem into a binary classification, the classes actually become more balanced and hence these three metrics perform more or less the same.
For the final results, precision and recall are both important metrics to consider. A poor precision indicates that the model would largely mis-predict YES for patients who would end up not being readmitted; this obviously wastes medical resources which could be used to help potentially readmitted patients. A poor recall indicates that the model would largely mis-predict NO for patients who would be readmitted eventually; this is also unacceptable because the patients' health is at risk. Financial considerations aside, the classification models that serve medical purposes usually prioritize on a good recall score. Our models perform poorly because they fail to raise the recall rate to over 0.60 for predicting YES. In practice, all the scores yield a similarly unsatisfactory score, which makes the choice of metric less compiling.

The Decision Tree and Random Forest models were fast to train. We were able to improve their performance easily but not beyond an accuracy of 0.64. As observed on the above Table 1, the recall rate is not acceptable for medical purposes and far below our expectation.

The Multi-Layer Perceptron and Support Vector Machine were much slower to train. Their performance were not improved after extensive hyperparameter optimization. As shown in Figure 2, the base model performs slightly better, but neither of the models yield satisfactory results. However, we notice that these models have higher recall rates as the number of correct predictions for YES is higher



(a) Base MLP Model Confusion Matrix. (b) Tuned MLP Model Confusion Matrix.

Figure 2: Compare Base and Tuned MLP Model with Confusion Matrices.

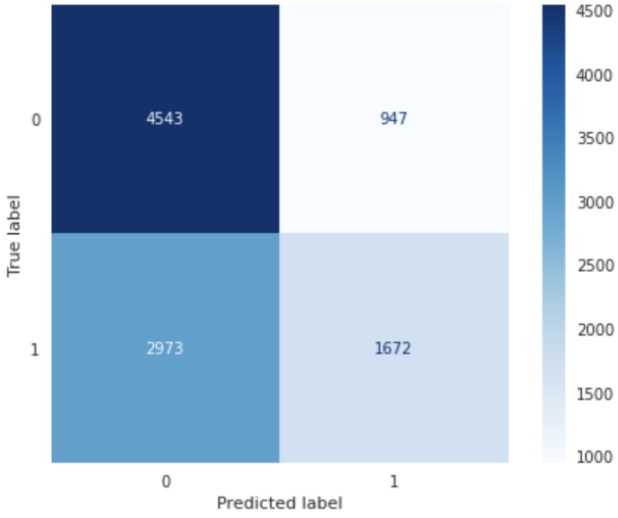


Figure 3: Best Random Forest Model Confusion Matrix.

than Figure 3. The SVM models show a similar confusion matrix asFigure 3.

3. Conclusions

In conclusion, Random Forest gives us the highest predicted score. Our analysis shows that the profile of readmission differed significantly in patients where "number_inpatient" is higher. We could achieve better results with more in-depth analysis and cleaning of the data and trying more parameters tuning. We also set an unrealistic goal of more than 90% accuracy of prediction without investigating the dataset we chose. Considering the large number of factors influencing patient health, we need more supporting data and domain expertise to tell which features are better indicators of these factors.

References

[1] scikit-learn: machine learning in python — scikit-learn 0.23.2 documentation. https://scikit-learn.org/stable/index.html.

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[3] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page I–115–I–123. JMLR.org, 2013.

[4] Rachel Draelos. Best use of train/val/test splits, with tips for medical data. https://glassboxmedicine.com/2019/09/15/best-use-of-train-val-test-splits-with-tips-for-medical-data/. Accessed: 2020-12-02.

[5] hyperopt. hyperopt/hyperopt-sklearn: Hyper-parameter optimization for sklearn. https://github.com/hyperopt/hyperopt-sklearn. Accessed: 2020-12-03.

[6] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25:2951–2959, 2012.

[7] Beata Strack, Jonathan P. DeShazo, Chris Gennings, Juan L. Olmo, Sebastian Ventura, Krzysztof J. Cios, and John N. Clore. Impact of hba1c measurement on hospital readmission rates: Analysis of 70,000 clinical database patient records. *BioMed Research International*, 2014:781670, Apr 2014.

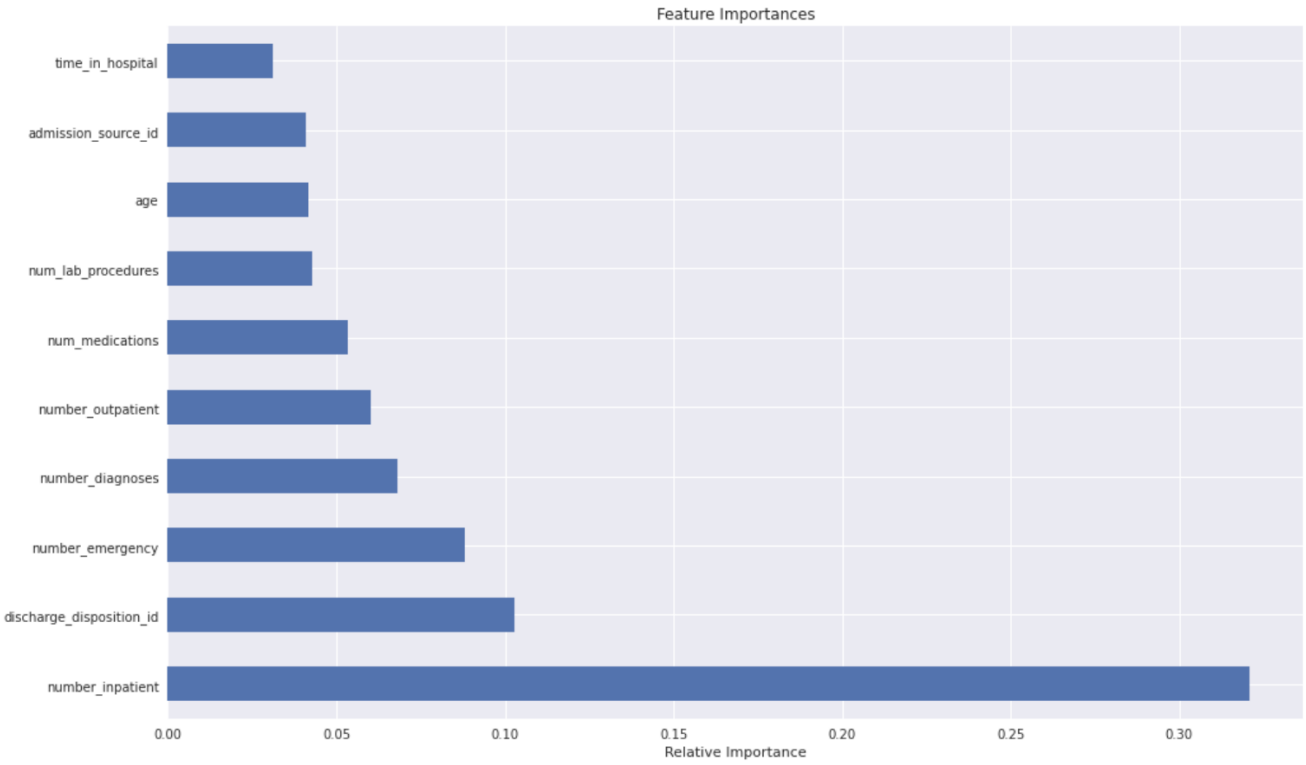


Figure 4: Top 10 Important Features as Assessed by the Random Forest Model.