CS 273A Project Report of Group 3 Hospital Readmission Prediction using Machine Learning Models on the Diabetes 130 US Hospitals Dataset (1999–2008)

Sihat Afnan, ID : 35227032
sihata@uci.edu
Dept. of Computer Science
UCI

Shagoto Rahman, ID : 65044968
shagotor@uci.edu
Dept. of Computer Science
UCI

Mahbub Raton, ID : 71099234
mraton@uci.edu
Dept. of Computer Science
UCI

Sihat Afnan, ID : 35227032, Shagoto Rahman, ID : 65044968, and Mahbub Raton, ID : 71099234. 2024. CS 273A Project Report of Group 3 Hospital Readmission Prediction using Machine Learning Models on the Diabetes 130 US Hospitals Dataset (1999–2008) . In . , 7 pages.

1 Introduction

Hospital readmissions pose a significant challenge to healthcare systems, leading to increased medical costs and reduced quality of care. Predicting readmission risks accurately can help hospitals implement targeted interventions, improving patient outcomes and resource allocation. This project explores machine learning approaches to predict hospital readmissions using the Diabetes 130-US Hospitals dataset, encompassing 101,766 patient encounters from 130 US hospitals between 1999 and 2008.

The dataset captures a diverse set of features, including patient demographics, medical conditions, treatments, and outcomes, with the target variable indicating readmission status. The analysis emphasizes the importance of demographic, clinical, and procedural factors in predicting readmissions. Advanced data preprocessing, feature engineering, and model development techniques were applied to handle data imbalances and improve predictive performance.

Through the implementation of Random Forest and Neural Network models, this project evaluates the strengths and limitations of each approach. The Random Forest model demonstrated robust performance with higher accuracy and stability after feature engineering and data balancing. On the other hand, the Neural Network model, while more sensitive to class imbalance, showed promise with improved predictions for minority classes after balancing techniques and hyperparameter tuning.

All three members contributed equally. Sihat Afnan worked on data exploration visualization and random forest model. Shagoto Rahman worked on the data preprocessing, exploration, visualization and on neural networks. Mahbub Raton assisted in data exploration.

2 Data Exploration and Visualization

2.1 Dataset Overview

The Diabetes 130-US Hospitals dataset consists of 101,766 patient encounters collected across 130 US hospitals between 1999 and 2008. It includes 50 features capturing patient demographics, medical conditions, treatments, and outcomes. The target variable, readmitted, indicates whether a patient was readmitted within 30 days (<30), beyond 30 days (>30), or not at all (NO).

2.2 Demographic feature Analysis: Age

The age distribution of patients indicates that the majority of patients belong to the [60-70] and [70-80] age groups. These groups represent a significant proportion of the dataset, with fewer patients in the younger age ranges (e.g., [0-10] and [10-20]).

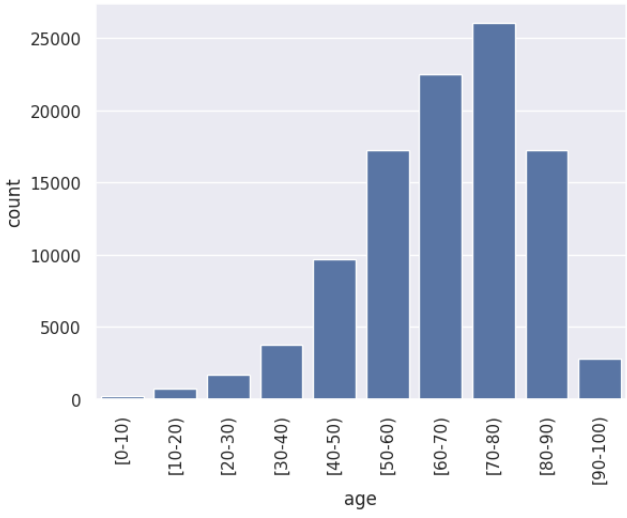

Figure 1. Distribution of Age

2.3 Demographic feature Analysis : Race

The distribution of the race variable indicates that the majority of patients in the dataset are Caucasian, followed by

African American. Other racial categories, including Asian, Hispanic, and Other, are underrepresented.

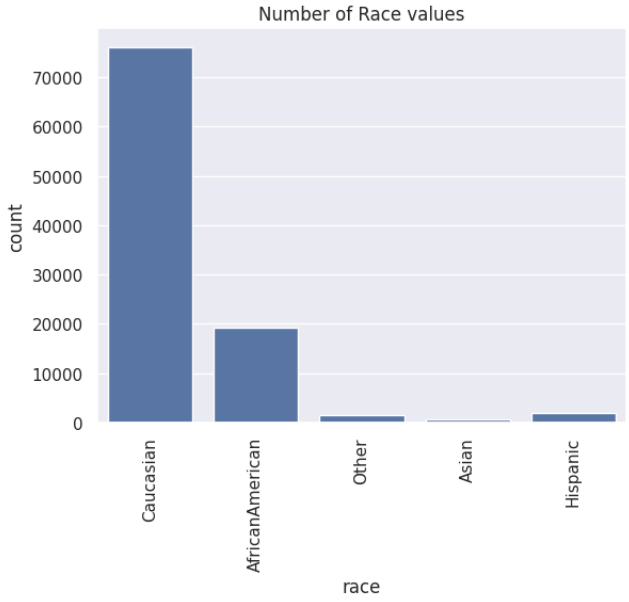

Figure 2. Distribution of Race

2.4 Clinical feature Analysis: Time in Hospital

The density plot compares the distribution of hospital stay durations (time_in_hospital) between patients who were readmitted and those who were not. Both distributions exhibit a similar pattern, with the majority of hospital stays being shorter (1-4 days). However, patients who were readmitted show slightly higher frequencies for longer hospital stays (5-10 days).

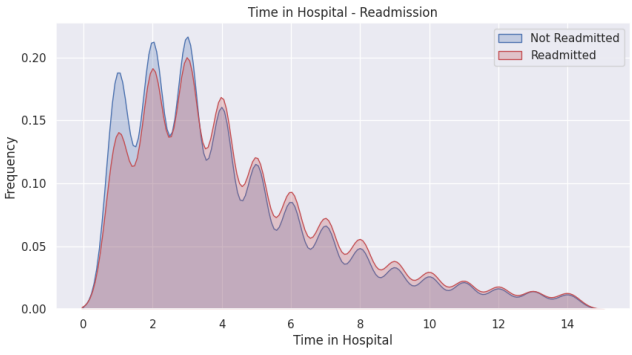

Figure 3. Density plot of stay duration

2.5 Clinical feature Analysis: Medication feature

The histogram shows the distribution of the number of medications prescribed to patients. The distribution is right-skewed, with most patients receiving fewer than 15 medications. A small subset of patients was prescribed a higher number of medications, with some receiving more than 40. This pattern suggests that the majority of patients have simpler treatment regimens, while a few require more complex medication plans.

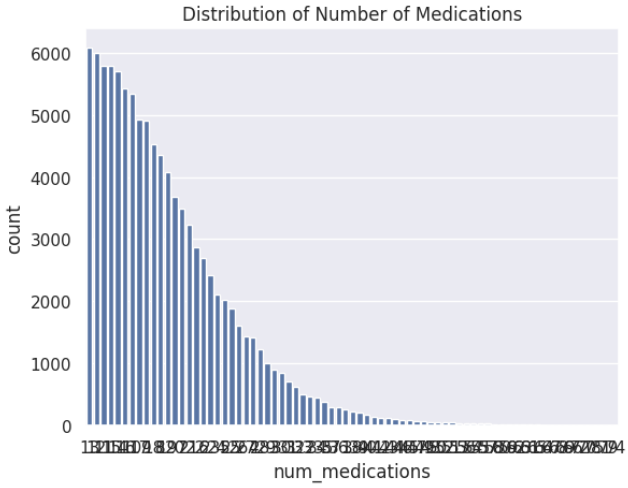

Figure 4. Histogram of number of medication

2.6 Clinical feature Analysis: Lab Procedures

The density plot illustrates the distribution of the number of lab procedures performed for patients who were readmitted and those who were not. Both distributions peak around 40 procedures, with readmitted patients showing a slightly higher frequency for larger numbers of lab procedures (50–70). This suggests a potential correlation between the number of lab procedures and readmission likelihood, where patients undergoing more extensive diagnostics may be at higher risk of readmission.

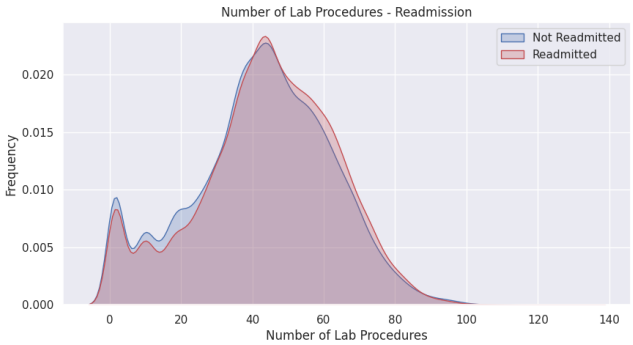

Figure 5. Density Plot of Lab Procedures

3 Models and Performance

Modifying the template — including but not limited to: adjusting margins, typeface sizes, line spacing, paragraph and

list definitions, and the use of the `\vspace` command to manually adjust the vertical spacing between elements of your work — is not allowed.

3.1 Random Forest

Random Forest is a versatile and robust machine learning algorithm, well-suited for classification tasks like predicting hospital readmissions. By combining multiple decision trees, Random Forest reduces overfitting and enhances generalization. In this section, we detail the application of Random Forest models to the Diabetes 130-US Hospitals dataset to predict patient readmissions.

3.1.1 Data Cleaning. The raw dataset contained various issues such as unused columns, missing values, undefined categories, and categorical variables that needed preprocessing. Below, we describe the detailed cleaning steps applied to prepare the data for modeling.

Dropping Irrelevant or Redundant Columns. The following columns were removed as they either contained IDs, constant values, or excessive missing values:

- `patient_nbr`: Unique identifier for patients, irrelevant for modeling.
- `citoglipton`, `examide`: Columns containing constant values.
- `weight`: Contains too many missing values (?).
- `encounter_id`: Unique encounter identifier, irrelevant for prediction.
- `diag_1`, `diag_2`, `diag_3`: Diagnostic codes, removed for simplicity.
- `payer_code`: Excessive missing values.

Handling Missing Values

- `race`: Missing values represented as ? were replaced with "Other" to ensure no loss of data.
- `medical_specialty`: Dropped due to an excessive number of missing values.

Encoding Categorical Variables

- `change`: Converted from `No` to 0 and `Ch` to 1.
- `diabetesMed`: Converted from `No` to 0 and `Yes` to 1.
- `readmitted`: Converted into numeric categories:
  - `NO`: 0
  - `<30`: 1
  - `>30`: 2

Removing Undefined or Invalid Rows

- Rows where `gender` was undefined (neither `"Male"` nor `"Female"`) were removed to clean the dataset.

Preparing Features and Target Variable

- **One-Hot Encoding**: Categorical variables were converted into dummy variables using one-hot encoding, dropping the first category to avoid multicollinearity.
- **Feature and Target Separation**:
  - `X`: Features for modeling.
  - `y`: Target variable (`readmitted`).

Then A **RandomForestClassifier with 20 estimators** was applied to the cleaned dataset to predict hospital readmissions. The dataset was split into training and testing sets with a 75%-25% ratio, stratifying by the target variable to preserve class proportions.

Class	Precision	Recall	F1-Score
0 (No Readmission)	0.60	0.80	0.69
1 (<30 Days)	0.27	0.04	0.07
2 (>30 Days)	0.46	0.35	0.40
Metric			**Value**
Macro Avg F1			0.38
Weighted Avg F1			0.52

Table 1. Performance Metrics for Readmission Classification

The result is not so good! Let's do some data balancing

3.1.2 Data Balancing. To address the class imbalance in the target variable, Synthetic Minority Oversampling Technique **(SMOTE)** was applied to the training data. This technique generates synthetic samples for minority classes to balance the dataset. The **Random Forest model was retrained using the balanced dataset** generated by SMOTE. The test data remained unchanged to evaluate generalization.

Class	Precision	Recall	F1-Score
0 (No Readmission)	0.61	0.75	0.67
1 (<30 Days)	0.24	0.09	0.13
2 (>30 Days)	0.45	0.38	0.41
Metric			**Value**
Macro Avg F1			0.41
Weighted Avg F1			0.52

Table 2. Performance Metrics for Readmission Classification with Data Balancing

While improvements were observed for minority classes (<30 Days and >30 Days), challenges remain due to overlapping features between classes.

3.1.3 Reducing Target Classes by Dropping readmitted > 30. The data was simplified by merging the readmitted > 30 category into the readmitted = 0 class (No Readmission). This was done to focus the analysis on distinguishing between patients readmitted within 30 days (<30) and those not readmitted.

Still, it's not improving much.Let's do some data engineering from scratch

Class	Precision	Recall	F1-Score
0 (No Readmission)	0.66	0.81	0.73
1 (<30 Days)	1.00	1.00	1.00
2 (>30 Days)	0.56	0.37	0.44
Metric			**Value**
Macro Avg F1			0.72
Weighted Avg F1			0.66
Accuracy			0.68

Table 3. Performance Metrics after readmittance>30 removed

3.1.4 Data Processing and Feature Engineering. The data was reprocessed from scratch to improve model performance by applying advanced data cleaning, transformation, and feature engineering techniques.

- **Dropping Irrelevant and Redundant Columns:**
  - Removed columns such as `admission_source_id`, `patient_nbr`, `citoglipton`, `weight`, `examide`, and `encounter_id` for being identifiers, constant, or irrelevant to modeling.
  - Dropped columns with excessive missing values, such as `payer_code`
- **Handling Missing and Ambiguous Values:**
  - Replaced missing values in `race` with `"Other"`.
  - Simplified the `medical_specialty` column by dropping it due to excessive missing data.
  - Filtered rows to keep only valid genders (`Male` and `Female`).
- **Encoding Target and Categorical Variables:**
  - Converted `readmitted` to numeric categories: 0 (`No`), 1 (`<30 days`), 2 (`>30 days`).
  - Encoded `change` and `diabetesMed` as binary variables.
- **Medication Dosage Transformation:**
  - Transformed medication columns (`metformin`, `insulin`, etc.) into numerical categories:
    - `No` or `Steady`: 0
    - `Up`: +1 (or +2 for `insulin`)
    - `Down`: -1 (or -2 for `insulin`)
  - Created a new feature `dosage_up_down` by summing the dosage change values for all medications.
- **Diagnosis Categorization:**
  - Converted diagnosis codes (`diag_1`, `diag_2`, `diag_3`) into one of 9 predefined categories using numeric and string-based conditions.
  - Replaced original diagnosis columns with categorized values.
- **Creating Aggregated Features:**
  - **Total Visits:** Combined `number_inpatient`, `number_outpatient`, and `number_emergency` into a single feature `total_visits`.
  - **Admission Type:** Encoded emergency admissions and combined them into a new admission feature.
  - **Total Procedures:** Summed `num_lab_procedures` and `num_procedures` into `total_procedures`.
- **Derived Indices:**
  - **LAE Index:** A composite index incorporating length of stay, admission acuity, and emergency visits.
  - **Leave Index:** A binary indicator for patients leaving against medical advice.
- **Lab and Test Results:**
  - Encoded `max_glu_serum` and `A1Cresult` values into numeric categories based on thresholds.
- **Age Grouping:**
  - Binned `age` into three categories: 0-30, 30-60, and 60-100 for simplification.
  - Combined age groups into new features and removed original `age` columns.
- **Diabetes Diagnosis Flag:**
  - Created a flag for diabetes-related diagnoses based on the three diagnosis columns.
- **Feature Engineering:**
  - Generated dummy variables for categorical features.
  - Selected the most relevant diagnostic category as the primary diagnosis.
- **Final Adjustments:**
  - Excluded rows with `readmitted = 2` (considered as No Readmission).
  - Prepared the final features (`X`) and target (`y`) for modeling.

The Random Forest model was retrained using the newly engineered dataset to evaluate the impact of the advanced data processing and feature engineering steps. The performance metrics on the test set after applying the improved data processing steps are as follows:

Class	Precision	Recall	F1-Score
0 (No Readmission)	0.89	0.98	0.93
1 (<30 Days)	0.13	0.02	0.04
Metric			**Value**
Accuracy			0.87
Macro Avg F1			0.49
Weighted Avg F1			0.83

Table 4. Performance Metrics for Readmission Classification

3.1.5 Significant Improvement. The overall accuracy increased to 87%, demonstrating the effectiveness of the advanced data processing steps. Precision and recall for the majority class (No Readmission) improved significantly. However, The model continues to struggle with identifying the <30 Days class, as indicated by low recall and F1-score. The improvements highlight the importance of thoughtful feature engineering in capturing relevant information for predictive modeling.

Analysis of Number of Estimators in Random Forest The impact of varying the number of estimators (n_estimators) in the Random Forest model was analyzed to identify the optimal number of trees for achieving maximum accuracy.

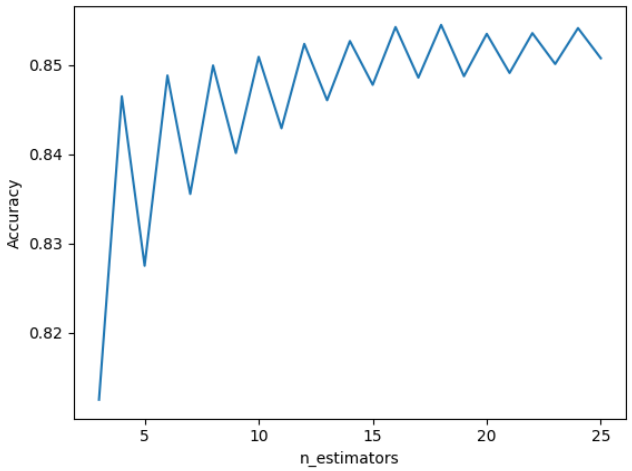

Figure 6. Accuracy vs Number of Estimators in Random Forest

From the plot, we see that at around n_estimator = 20, the model somewhat converges and doesn't improve further.

3.2 Neural Network

3.2.1 Missing Values. First of all, the indices are cleaned where all diagnostic-related attributes were missing, namely 'diag_1', 'diag_2', 'diag_3'. In addition, data were also filtered where the gender value was "unknown". The next part was to remove the columns with the most missing values. Table 5 shows the columns with most missing values. Columns with a threshold of 30% missing data were removed.

Feature	Missing Value Count
weight	98,568
max_glu_serum	96,419
A1Cresult	84,747
medical_specialty	49,949
payer_code	40,255
race	2,273
diag_3	1,422
diag_2	357
diag_1	20

Table 5. Missing Values in Dataset

3.2.2 Missing Value Imputation. Now the job is to impute values for other missing values. Since the dataset has both numerical and categorical features, the imputations were handled separately. Categorical columns were identified with a datatype object that defines either text or categorical data. In case of the categorical data, we imputed missing values with most frequent data. SimpleImputer class was utilized in this case and this step ensures that the imputed values align with the distribution of the data and maintain the categorical nature.

Next, for numerical columns, the imputation strategy enabled here is the 'mean' strategy. This is accomplished by utilizing another SimpleImputer instance configured with the 'mean' strategy. It computes the average of the existing values in each numerical column and fills in any missing entries with this calculated mean. These steps collectively ensure that the data set is free from missing values, making it ready for further analysis or machine learning applications without introducing substantial bias.

3.2.3 Encoding. To handle categorical data, they were transformed into numerical values using label encoding technique by enabling categorical values to be defined by unique integers. Transforming categorical features into a machine-readable format, each categorical column in the dataset is processed independently, ensuring an efficient method for managing multiple columns. For numerical features, we normalize all values to ensure they are on a consistent scale. The StandardScaler standardizes the data by centering it around the mean and scaling it to have unit variance, preventing features with larger ranges from disproportionately influencing the model. The target variable is encoded manually. A dictionary is used to assign numeric labels to the target values: both 'NO' and '>30' are mapped to 0, while '<30' is mapped to 1. This process converts the target into a binary format ideal for binary classification tasks. Figure 7 shows a chunk of data after encoding.

```
X[categorical_columns].head(5)
```

	race	gender	age	diag_1	diag_2	diag_3	metformin	repaglinide	nateglinide	chlorpropamide	...	examide	citoglipton	insulin	glyburide-metformin	glipizide-metformin
0	2	0	0	124	748	789	1	1	1	1	...	0	0	1	1	0
1	2	0	1	143	79	121	1	1	1	1	...	0	0	3	1	0
2	0	0	2	454	78	766	1	1	1	1	...	0	0	1	1	0
3	2	1	3	554	97	248	1	1	1	1	...	0	0	3	1	0
4	2	1	4	54	24	86	1	1	1	1	...	0	0	2	1	0

5 rows × 31 columns

Figure 7. Data after encoding.

3.2.4 Neural Network Experiments, Results and Discussion. The data is divided into 3 sets: training, validation and testing. Here, Two variations are created. First training 80%, validation: 10%, testing: 10%. Another training 70%, validation: 15%, testing: 15%. Since both the variants show similar results we only show the first one. We started our experiment with a very simple neural network model (Figure 8) with three fully connected layers with relu activation only. We trained 100 epochs of the model with a batch size of 128, learning rate 0.0001 and in figure 9(a) we can see that the model is over-fitting as the training loss decreases but

the test loss increases and the accuracy is 0.8889. However, problem is this model always predicts 0 and thus this class related result does not hint at good accuracy.

```
class DiabetesNN(nn.Module):
    def __init__(self, input_dim, num_classes=2):
        super(DiabetesNN, self).__init__()
        self.fc1 = nn.Linear(input_dim, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, num_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Figure 8. Simple Neural Network

To make the model better and since the data is highly imbalanced, we introduce balancing the class weights. The "balanced" option is used to automatically calculate class weights that are inversely proportional to the frequency of each class in training values of y, preventing the model from favoring the majority class. These class weights are stored in a dictionary, with each class label as a key and its corresponding weight as the value. The pos_weight parameter in the BCEWithLogitsLoss function is set to the weight of the positive class (class 1), allowing the loss function to give more importance to the positive class during training, which is especially useful for imbalanced datasets. The pos_weight is provided as a tensor to the loss function to make it more sensitive to class 1. For that the result is better as we can see in Figure 9(b).

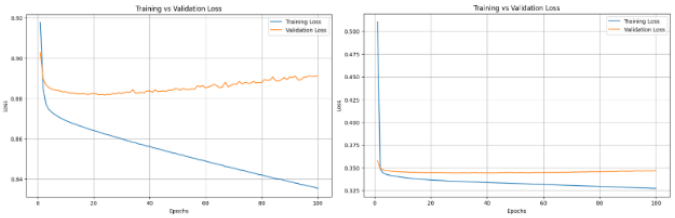

Figure 9. a. Without class weight loss, b. With class weight loss.

Now the models' scores are better because now it is more confident with other classes and we got almost the same precision, recall, f1 like the previous but now it is not only predicting 0 but also it is predicting 1 and overall score is good. But again this model also has over-fitting. So to reduce the over-fitting the first thing we tried is early stopping and dropout because both early stopping and dropout will block the training process from memorizing the data. We can see the results in Figure 10. Indeed, the gap between training and evaluation loss decreases.

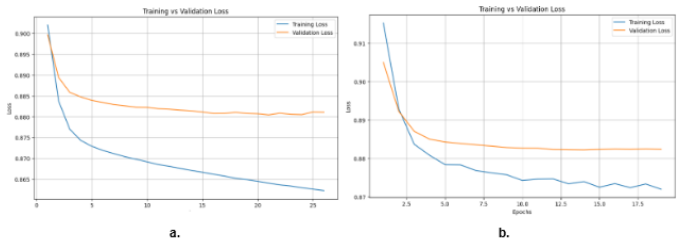

Figure 10. a. Early Stopping, b. Dropout.

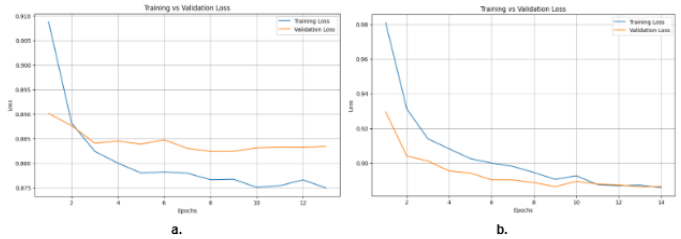

Figure 11. a. Batch Norm, b. Weight Decay.

Moreover we did experiments with batchnorm layers and weight decay to further reduce overfitting. Figure 11 represents the loss of the models and we can see that the gap between train and test loss decreases.

Model	Learning Rate (lr)	Batch Size	Dropout Rate	Hidden Units	Weight Decay
1	0.001	64	0.2	64	0.0001
2	0.0005	128	0.3	128	0.00001
3	0.01	128	0.4	256	0.001
4	0.0001	64	0.5	512	0.0005
5	0.001	64	0.6	128	0.0001
6	0.001	256	0.2	128	0.0001

Table 6. Simple NN Model Configuration

Model	Loss	Accuracy	Precision	Recall	F1 Score
1	0.880	0.850	0.827	0.850	0.837
2	0.882	0.838	0.825	0.838	0.831
3	0.887	0.889	0.841	0.889	0.838
4	0.880	0.860	0.823	0.860	0.839
5	0.881	0.857	0.821	0.857	0.837
6	0.881	0.827	0.825	0.827	0.826

Table 7. Model Performance Metrics

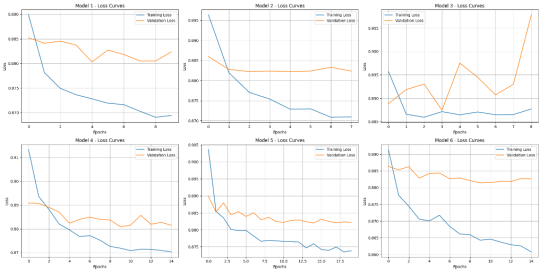

Figure 12. All model loss results

```python
class CNNModel(nn.Module):
    def __init__(self, input_dim, filters=64, dropout_rate=0.4, num_classes=1):
        super(CNNModel, self).__init__()

        # Conv Layers
        self.conv1 = nn.Conv1d(in_channels=1, out_channels=filters, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool1d(kernel_size=2, stride=2, padding=0)
        self.conv2 = nn.Conv1d(in_channels=filters, out_channels=filters*2, kernel_size=3, stride=1, padding=1)

        # Fully connected layers
        self.fc1 = nn.Linear(filters * 2 * self.feature_map_size, 512)  # Input size to fc1 dynamically calculated
        self.fc2 = nn.Linear(512, num_classes)

        # Activation and dropout
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=dropout_rate)


    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(x)
        x = self.relu(self.conv2(x))
        x = self.pool(x)
        x = x.view(x.size(0), -1)  # Flatten for fully connected layer
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

Figure 13. Simple CNN Model

Lastly we experimented with the hyperparameters to compare the different configurations of the model. Table 6 and Table 7 shows the configurations and the results of the models respoectively. For all the experiments, the tolerance was used 5. Figure 12 shows the loss analysis of the models. Model 2 and 6 were the top performers in this case for results and class balancing as all of them got similar scores but 2 and 6 got the best confusion matrix and class balance.

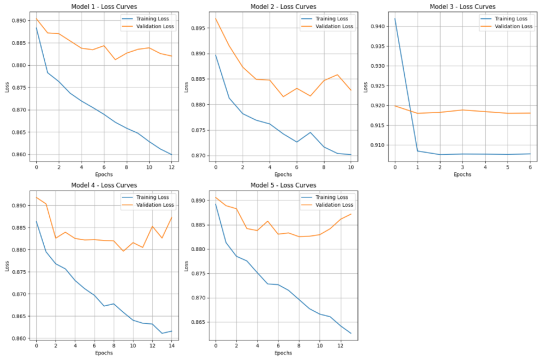

Figure 14. CNN model loss results

We did another example with CNN model to see the response. Figure 13 shows a simple CNN model. So we created a bunch of configurations with the model and Table 8 and Table 9 show the configurations and results of those models respectively. From the table results and also Figure 14 we can see that among all these experiments, model 2 did the best as model 1 always predicted class 0, other models apart from 2 show poor scores to over compensate for class imbalance and as a result learn to predict more 1s. But model 2's result were balanced we got good overall scores and better confusion matrix with balanced class prediction.

Model	LR	Batch Size	Dropout Rate	Filters	Hidden Units	Weight Decay
1	0.0001	64	0.2	32	128	1e-5
2	0.001	128	0.4	64	256	1e-4
3	0.01	128	0.5	128	512	0
4	0.0005	256	0.3	64	128	1e-4
5	0.001	64	0.4	32	256	1e-5

Table 8. CNN Model Configurations

Model	Accuracy	Precision	Recall	F1 Score
1	0.889162	0.790774	0.889162	0.837087
2	0.848777	0.80147	0.848777	0.823127
3	0.164292	0.780199	0.164292	0.138939
4	0.393829	0.805832	0.393829	0.479301
5	0.587993	0.808385	0.587993	0.664589

Table 9. Model Performance Metrics

4 Conclusion

In this project, we explored the Diabetes 130-US Hospitals dataset, starting with data exploration to analyze the demographic and clinical features through visualizations that highlighted their distributions and patterns. We applied both machine learning and deep learning approaches: Random Forest for both three-class and binary classification and neural networks for binary classification. Data preprocessing, balancing, and feature engineering were key to preparing the dataset for each approach. For Random Forest, preprocessing steps significantly improved performance, and we identified the optimal number of estimators. In the neural network experiments, we tackled challenges like class imbalance and overfitting using techniques such as class weighting, early stopping, dropout layers, and weight decay, which led to better model performance. The project highlights the importance of tailored preprocessing and algorithm-specific strategies in achieving reliable predictions.