Machine Learning : Project Report

Harsh Kumar
IMT2021016

Subhajeet Lahiri
IMT2021022

Sai Madhavan G
IMT2021101

***Abstract*—This technical report presents the implementation and results of an ensemble-based approach for the Canadian Hospital Re-admittance Challenge. The project was carried out by Harsh Kumar (IMT2021016), Subhajeet Lahiri (IMT2021022), and Sai Madhavan G (IMT2021101). The primary objective was to predict hospital readmission for patients using non-neural ensemble models. The data used in this report has undergone preprocessing steps outlined in the preprocessing.py file.**

Link to the GitHub Repository

I. Pre-processing

A. *Structure of the Data*

The training data consisted of 71236 rows, encompassing 49 features and one target variable - readmission_id. Of the 49 features:

- 10 were categorical variables: *'race', 'gender', 'admission_type_id', 'discharge_disposition_id', 'admission_source_id', 'payer_code', 'medical_specialty', 'diag_1', 'diag_2', 'diag_3', 'max_glu_serum'*
- 25 were categorical variables related to dosages of specific drugs : *'metformin', 'repaglinide', 'nateglinide', 'chlorpropamide', 'glimepiride', 'acetohexamide', 'glipizide', 'glyburide', 'tolbutamide', 'pioglitazone', 'rosiglitazone', 'acarbose', 'miglitol', 'troglitazone', 'tolazamide', 'examide', 'citoglipton', 'insulin', 'glyburide-metformin', 'glipizide-metformin', 'glimepiride-pioglitazone', 'metformin-rosiglitazone', 'metformin-pioglitazone', 'change', 'diabetesMed'*
- 11 were either quantitative variables or could be reduced to one: *'age', 'weight', 'time_in_hospital', 'num_lab_procedures', 'num_procedures', 'num_medications', 'number_outpatient', 'number_emergency', 'number_inpatient', 'max_glu_serum', 'A1Cresult'*
- 2 were identifiers: *'enc_id' and 'patient_id'*

B. *Feature Distributions*

Statistical descriptors of some quantitative features pre-cleaning are presented in Table I. Table II shows the percentage of null values in each column. Fig I-C - I-C plots the data distribution of each feature. For categorical variables, we have used bar charts denoting the frequency of each value. For quantitative variables, we have used line plots.

C. *Feature Relationships*

Relationships with the target variable (readmission_id) were explored using various visualization techniques, such as stacked bar plots for categorical features and boxplots for quantitative variables categorized by the target. Bar charts for the top 10 most common values in diag_x were utilized.

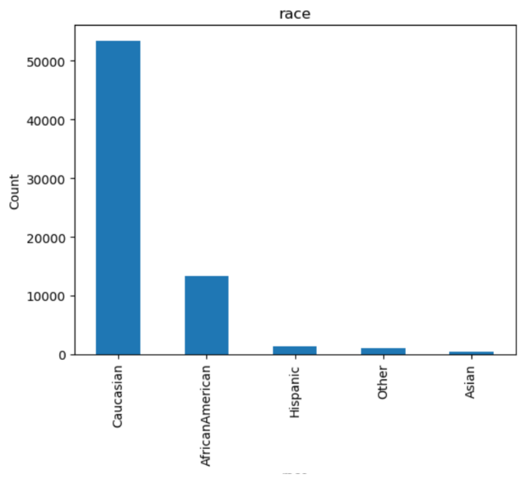
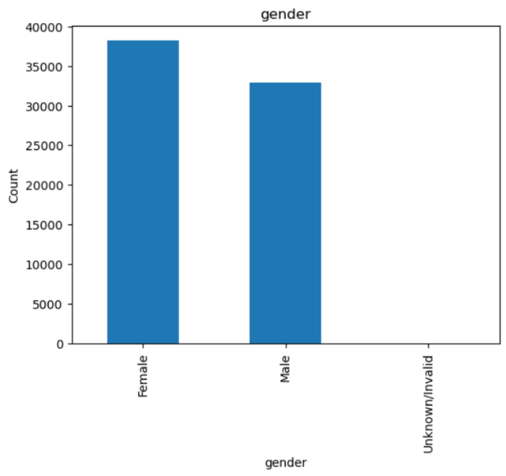


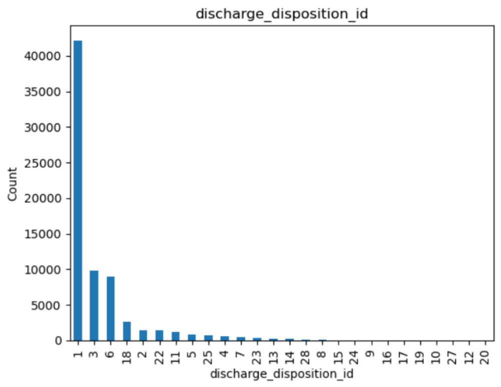
TABLE I
STATISTICAL DESCRIPTORS OF QUANTITATIVE FEATURES

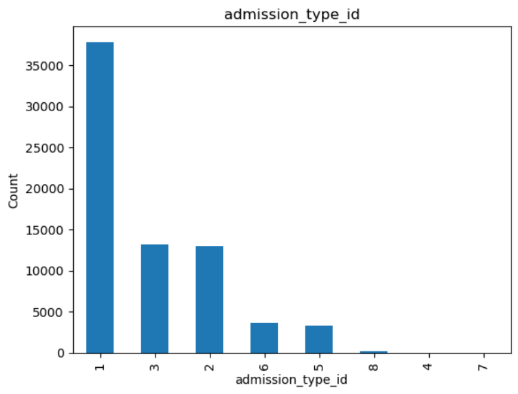
Feature	Count	Mean	Std	Min	25%	50%	75%	max
`time_in_hospital`	71236	4.3946	2.9799	1.0000	2.0000	4.0000	6.0000	14.0000
`num_lab_procedures`	71236	43.1126	19.6588	1.0000	31.0000	44.0000	57.0000	132.0000
`num_procedures`	71236	1.3425	1.7055	0.0000	0.0000	1.0000	2.0000	6.0000
`num_medications`	71236	16.0164	8.1308	1.0000	10.0000	15.0000	20.0000	81.0000
`number_outpatient`	71236	0.3711	1.2699	0.0000	0.0000	0.0000	0.0000	42.0000
`number_emergency`	71236	0.1958	0.8645	0.0000	0.0000	0.0000	0.0000	63.0000
`number_inpatient`	71236	0.6353	1.2692	0.0000	0.0000	0.0000	1.0000	19.0000

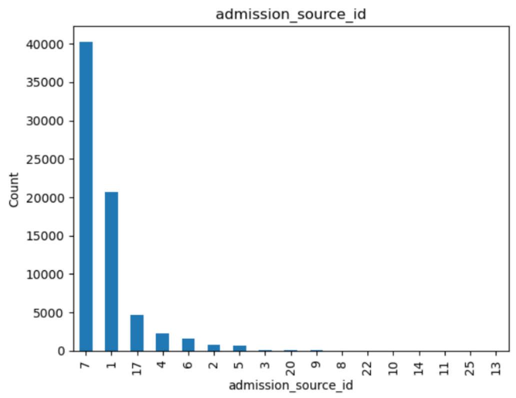
TABLE II
PERCENTAGE OF NULL VALUES IN COLUMNS

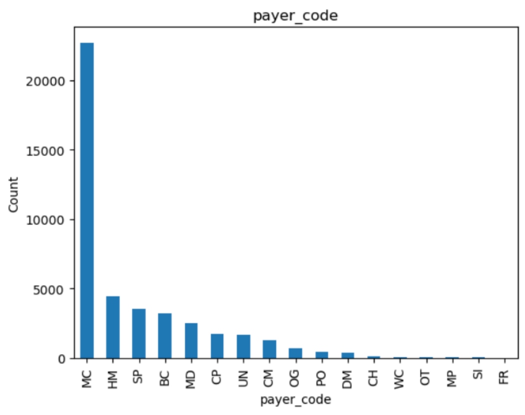
Column	Null %
`weight`	96.841%
`max_glu_serum`	94.776%
`A1Cresult`	83.323%
`medical_specialty`	49.034%
`payer_code`	39.556%
`race`	2.276%
`diag_3`	1.388%
`diag_2`	0.343%
`diag_1`	0.021%

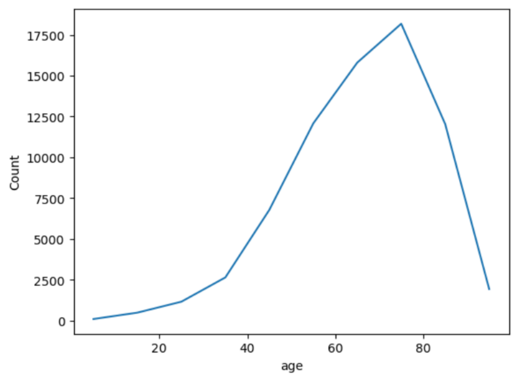


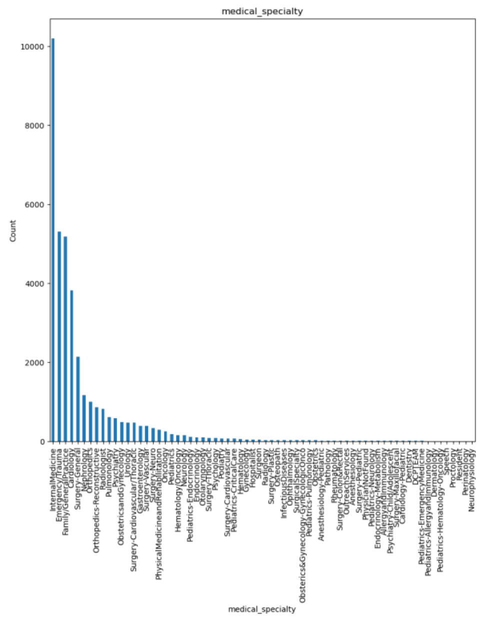


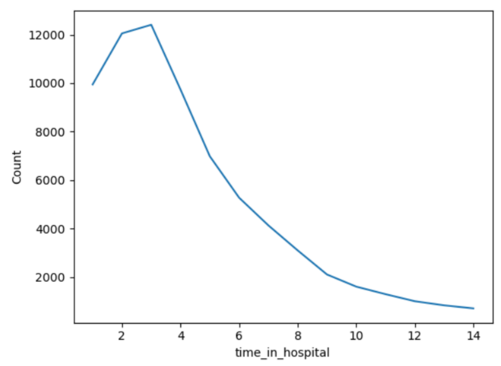


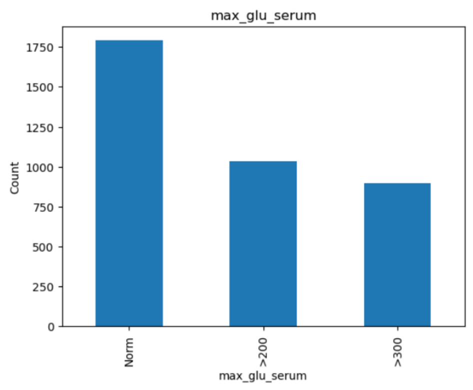


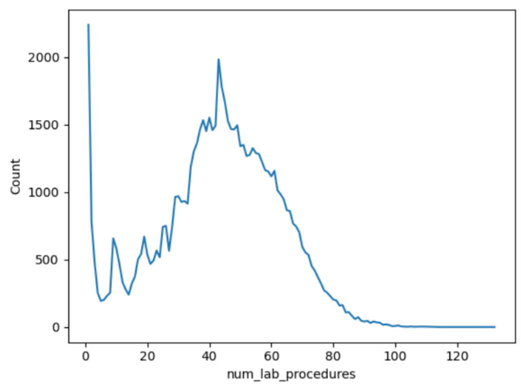

payer_code
Count
20000
15000
10000
5000
0
MC
HM
SP
BC
MD
CP
UN
CM
OG
PO
DM
CH
WC
OT
MP
SI
FR
payer_code

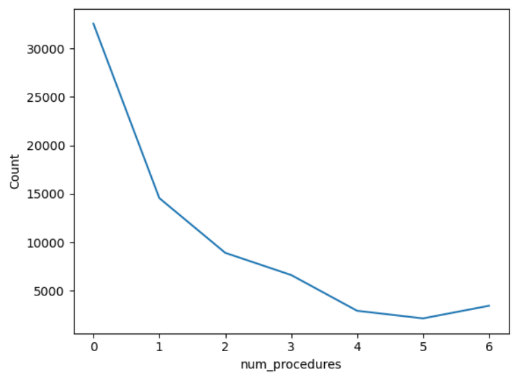

17500
15000
12500
10000
7500
5000
2500
0
Count
20
40
60
80
age

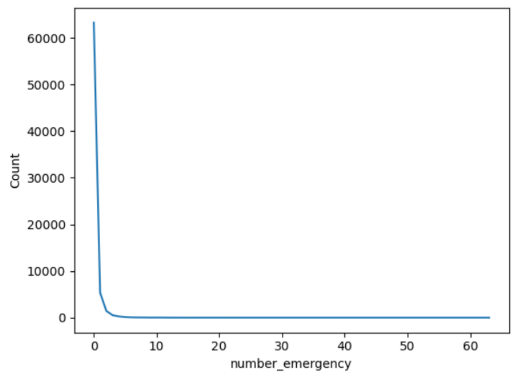

medical_specialty
10000
8000
6000
4000
2000
0
Count
InternalMedicine
Emergency/Trauma
Family/GeneralPractice
Cardiology
Surgery-General
Nephrology
Orthopedics
Orthopedics-Reconstructive
Radiologist
Pulmonology
Psychiatry
ObstetricsandGynecology
Urology
Surgery-Cardiovascular/Thoracic
Gastroenterology
Surgery-Vascular
Surgery-Neuro
PhysicalMedicineandRehabilitation
Oncology
Pediatrics
Hematology/Oncology
Neurology
Pediatrics-Endocrinology
Endocrinology
Otolaryngology
Surgery-Thoracic
Psychology
Podiatry
Surgery-Cardiovascular
Pediatrics-CriticalCare
Hematology
Gynecology
Hospitalist
Surgeon
Radiology
Surgery-Plastic
Osteopath
InfectiousDiseases
Ophthalmology
SurgicalSpecialty
Obsterics&Gynecology-GynecologicOnco
Pediatrics-Pulmonology
Obstetrics
Anesthesiology-Pediatric
Pathology
Rheumatology
Surgery-Colon&Rectal
OutreachServices
Anesthesiology
Surgery-Pediatric
PhysicianNotFound
Pediatrics-Neurology
Endocrinology-Metabolism
AllergyandImmunology
Psychiatry-Child/Adolescent
Surgery-Maxillofacial
Cardiology-Pediatric
Dentistry
DCPTEAM
Pediatrics-EmergencyMedicine
Pediatrics-AllergyandImmunology
Dermatology
Pediatrics-Hematology-Oncology
Speech
Proctology
Resident
Perinatology
Neurophysiology
medical_specialty

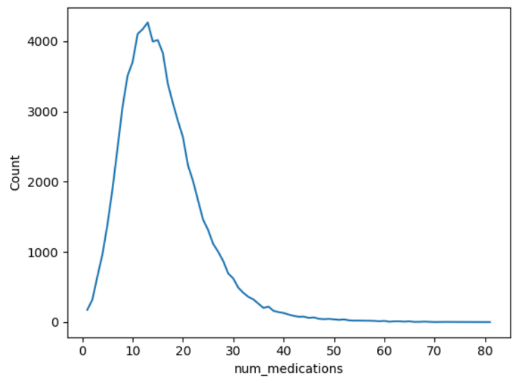

12000
10000
8000
6000
4000
2000
Count
2
4
6
8
10
12
14
time_in_hospital

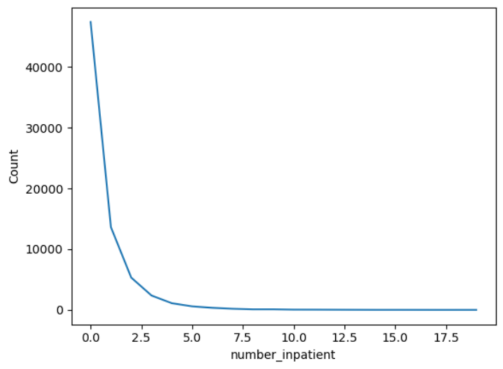

max_glu_serum
1750
1500
1250
1000
750
500
250
0
Count
Norm
>200
>300
max_glu_serum

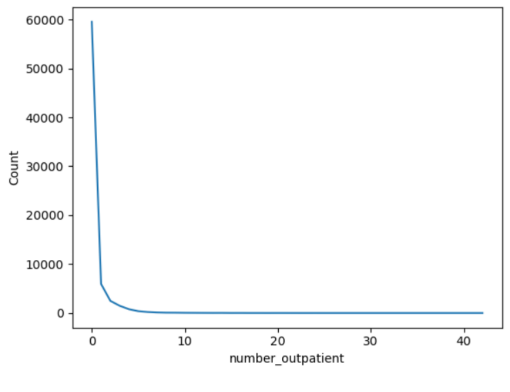

2000
1500
1000
500
0
Count
0
20
40
60
80
100
120
num_lab_procedures

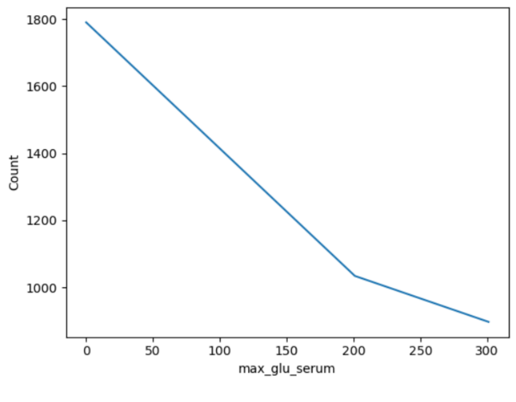

Count
num_procedures

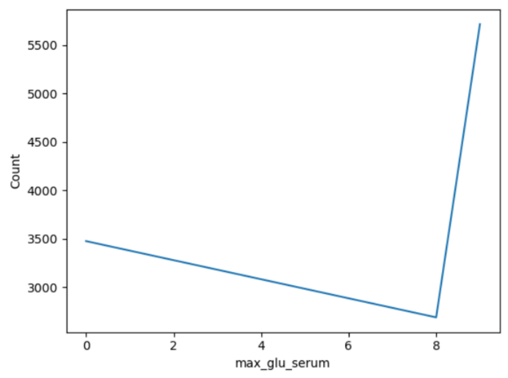

Count
number_emergency

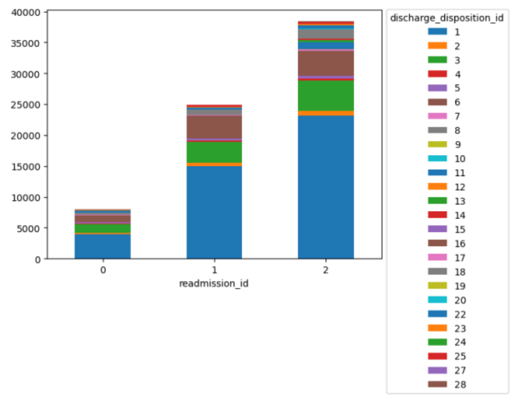

Count
num_medications


Count
number_inpatient


Count
number_outpatient

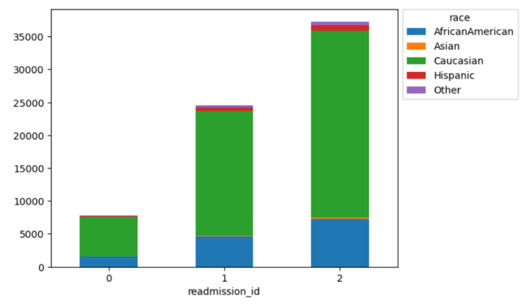

Count
max_glu_serum

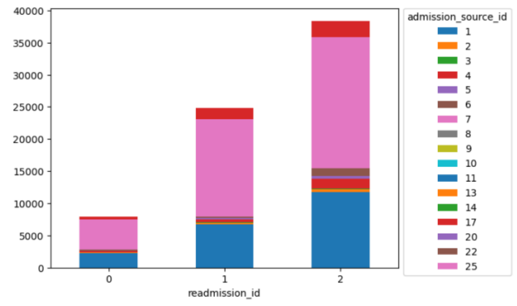


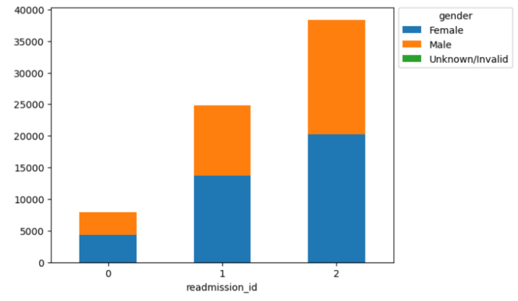


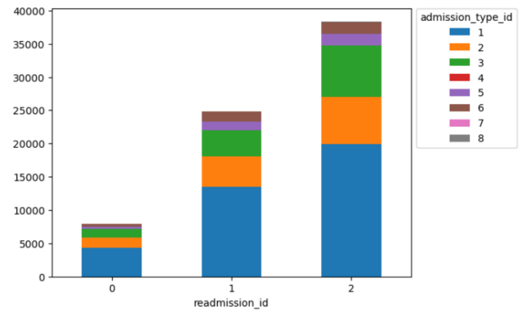
D. Feature Relationships among Themselves

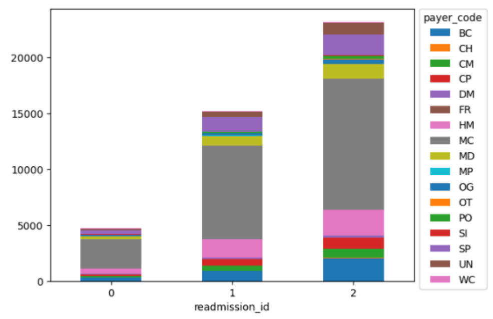
We can visualize the correlation between the target (which is a categorical variable) and other categorical variables using stacked bar plots and that between the target and quantitative variables using boxplots for each category. (Fig I-D - I-D)

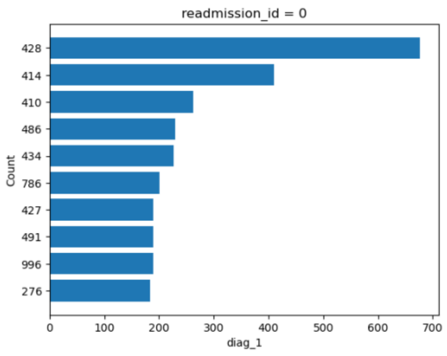


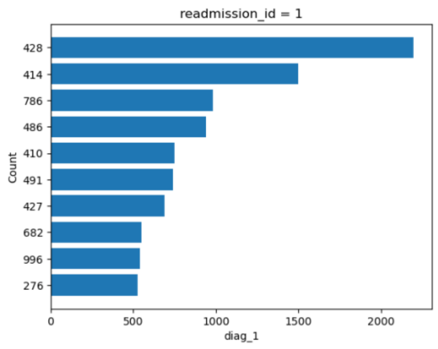


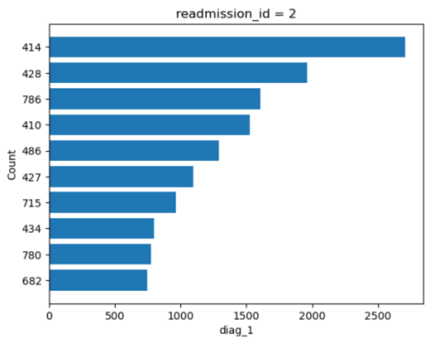


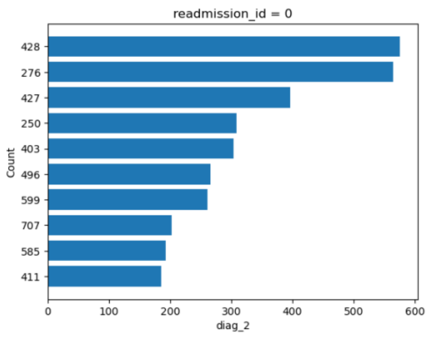


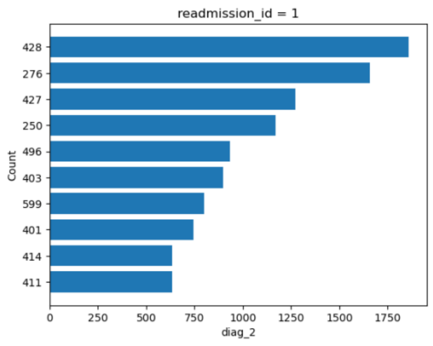


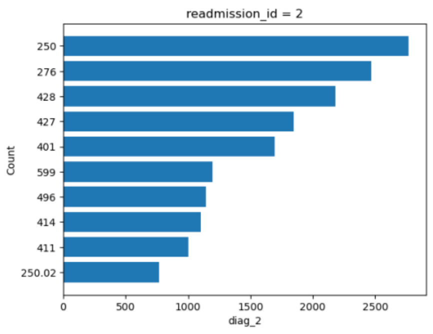

readmission_id = 0
428
414
410
486
434
786
427
491
996
276
Count
0
100
200
300
400
500
600
700
diag_1

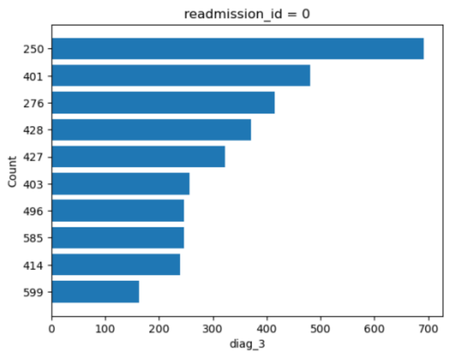

readmission_id = 1
428
414
786
486
410
491
427
682
996
276
Count
0
500
1000
1500
2000
diag_1

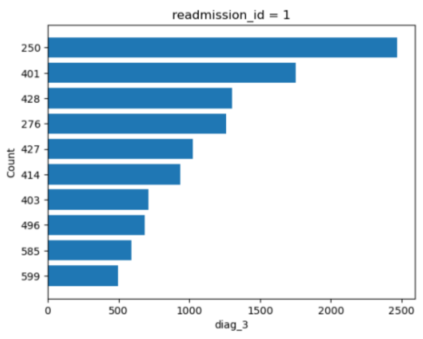

readmission_id = 2
414
428
786
410
486
427
715
434
780
682
Count
0
500
1000
1500
2000
2500
diag_1

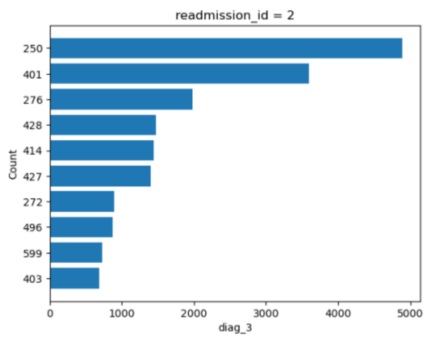

readmission_id = 0
428
276
427
250
403
496
599
707
585
411
Count
0
100
200
300
400
500
600
diag_2

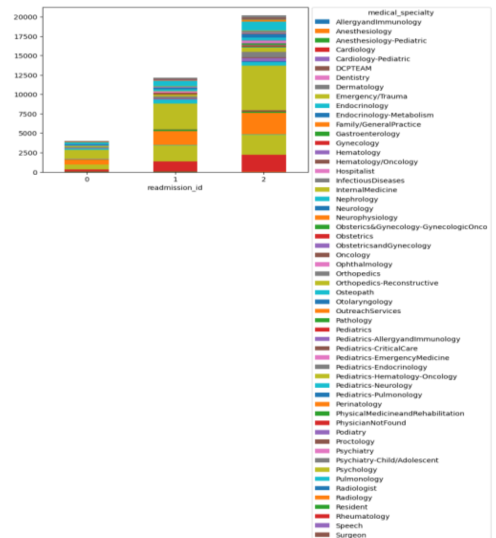

readmission_id = 1
428
276
427
250
496
403
599
401
414
411
Count
0
250
500
750
1000
1250
1500
1750
diag_2

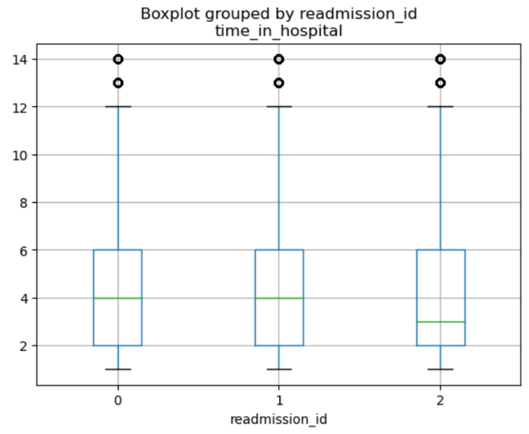

readmission_id = 2
250
276
428
427
401
599
496
414
411
250.02
Count
0
500
1000
1500
2000
2500
diag_2

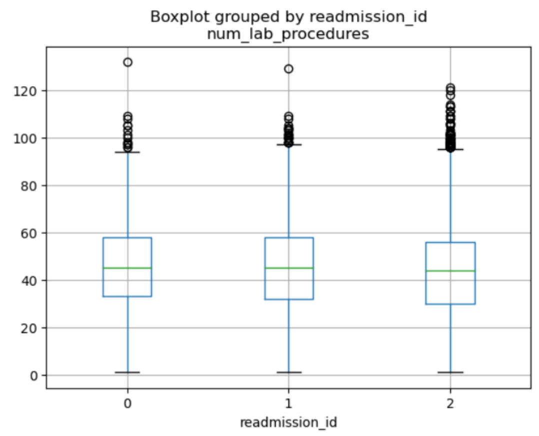

readmission_id = 0
250
401
276
428
427
403
496
585
414
599
Count
0
100
200
300
400
500
600
700
diag_3

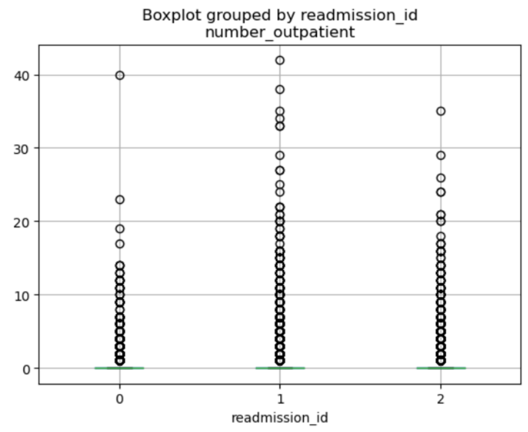

readmission_id = 1
250
401
428
276
427
414
403
496
585
599
Count
0
500
1000
1500
2000
2500
diag_3

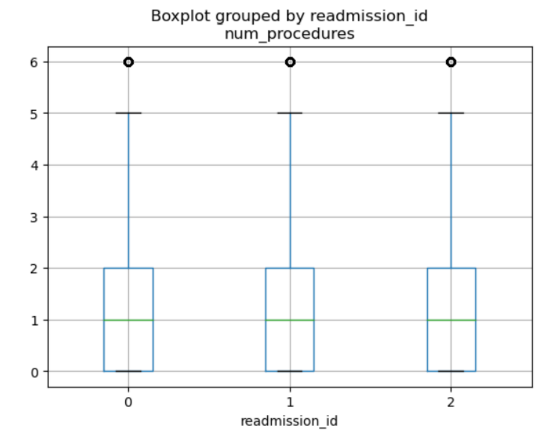

readmission_id = 2
250
401
276
428
414
427
272
496
599
403
Count
0
1000
2000
3000
4000
5000
diag_3

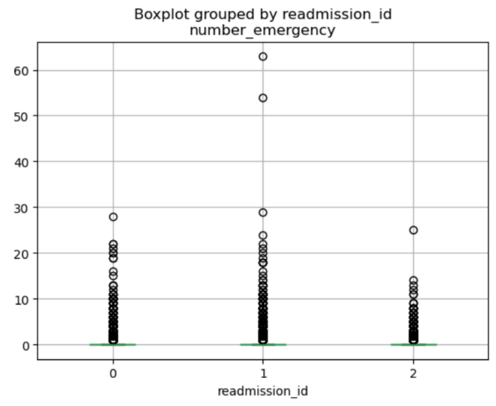

20000
17500
15000
12500
10000
7500
5000
2500
0
0
1
2
readmission_id
medical_specialty
AllergyandImmunology
Anesthesiology
Anesthesiology-Pediatric
Cardiology
Cardiology-Pediatric
DCPTEAM
Dentistry
Dermatology
Emergency/Trauma
Endocrinology
Endocrinology-Metabolism
Family/GeneralPractice
Gastroenterology
Gynecology
Hematology
Hematology/Oncology
Hospitalist
InfectiousDiseases
InternalMedicine
Nephrology
Neurology
Neurophysiology
Obsterics&Gynecology-GynecologicOnco
Obstetrics
ObstetricsandGynecology
Oncology
Ophthalmology
Orthopedics
Orthopedics-Reconstructive
Osteopath
Otolaryngology
OutreachServices
Pathology
Pediatrics
Pediatrics-AllergyandImmunology
Pediatrics-CriticalCare
Pediatrics-EmergencyMedicine
Pediatrics-Endocrinology
Pediatrics-Hematology-Oncology
Pediatrics-Neurology
Pediatrics-Pulmonology
Perinatology
PhysicalMedicineandRehabilitation
PhysicianNotFound
Podiatry
Proctology
Psychiatry
Psychiatry-Child/Adolescent
Psychology
Pulmonology
Radiologist
Radiology
Resident
Rheumatology
Speech
Surgeon

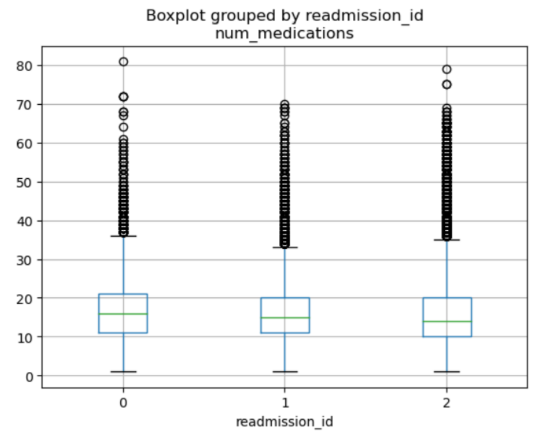

Boxplot grouped by readmission_id
time_in_hospital
14
12
10
8
6
4
2
0
1
2
readmission_id

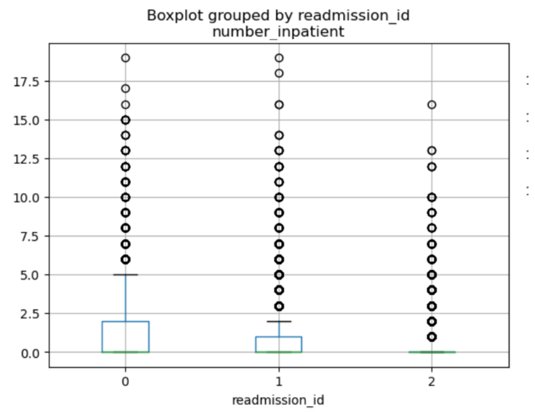

Boxplot grouped by readmission_id
num_lab_procedures
readmission_id

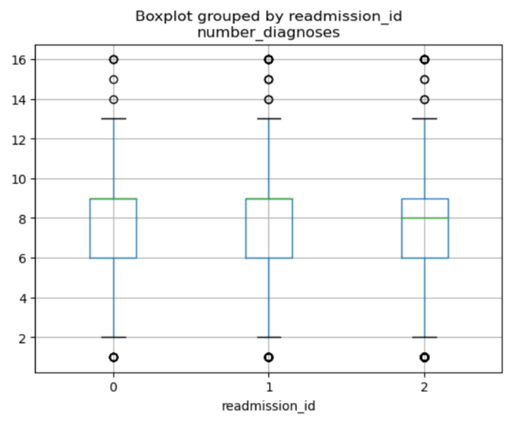

Boxplot grouped by readmission_id
number_outpatient
readmission_id

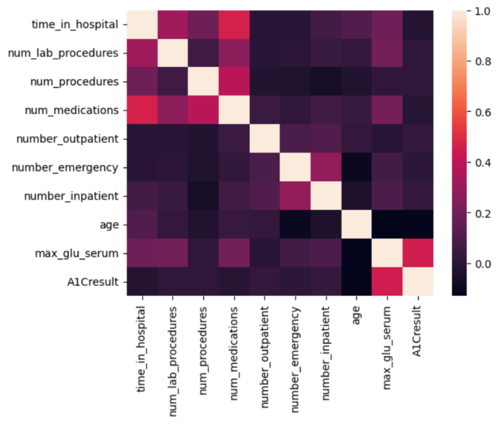

Boxplot grouped by readmission_id
num_procedures
readmission_id


Boxplot grouped by readmission_id
number_emergency
readmission_id


Boxplot grouped by readmission_id
num_medications
readmission_id


Boxplot grouped by readmission_id
number_inpatient
readmission_id





For diag_x, we can have bar charts for the 10 most common values. (Fig I-D - I-D)

We observe a mostly cold heatmap (Fig I-D)

E. Preprocessing and Feature Extraction

*1) **Data Cleaning**:* Strategies for handling null values were implemented:

1) Dropping columns: ‘weight‘, ‘max_glu_serum‘, ‘A1Cresult‘
2) Imputing constant value: ‘medical_specialty‘, ‘payer_code‘
3) Imputing the mode: ‘race‘, ‘diag_x‘

Although we chose these methods purely based on which improved our score, we have a few hypotheses on why they worked well.

- For ‘weight‘, ‘max_glu_serum‘ and ‘A1Cresult‘, dropping them entirely made sense since upwards of 80imputation for this column did not affect the score as much since the effective amount of information being conveyed is less
- For ‘medical_specialty‘ and ‘payer_code‘, although the number of null values is significant, the amount of information in the non-null rows was significant enough that introducing new categories to denote null did not adversely affect the score.
- The ‘race‘ and ‘diag‘ columns had negligible null values so it made no sense to drop them. On trying the different imputation methods available, the difference in score each of them caused was miniscule since there were very few null values to begin with. So, we went with the simplest approach of imputing with the mode.

F. Encoding

Two encoding methods were experimented with - one-hot encoding and ordinal encoding..

G. Scaling and Normalizing

Scaling and normalization were applied to some quantitative features but did not significantly impact the model's performance, likely due to the presence of outliers and the model binning quantitative data into discrete bins.

H. Methods for Improving Feature Quality

Feature engineering was employed to enhance the model's performance:

- **Drugs**:There were 23 categorical columns mentioning whether that drug has been increased, decreased, not changed, or not prescribed. Although these columns would carry much information on how the patient was diagnosed and treated, removing the column had little drop in performance. We assumed this was the case since the data was very sparse with very few cells having meaningful data. To reduce the complexity and number of columns, we consolidate the changes in drug dosages for a particular encounter in terms of number of drugs increased, decreased, or not changed in dosage. We created three new columns to represent this and dropped the original 23 columns. This led to a one percent increase in the score.
- **Diagnoses**: We anticipated the three ‘diag‘ columns to be pretty important as they contained the ICD9 codes of the diseases they were diagnosed with. However, removing them didn't really affect the score as much. We attributed this to the large number of unique categories in the columns (around 800 in each column). So, in order to decrease the complexity, we grouped similar diseases into same category using their ICD9 codes with the help of the ‘icd9cms‘ library. This reduced the number of unique categories from around 800 to only 120 and improved performance by around 0.5%.
- **Number of patient visits:** We noticed a lot of repetition in the ‘patient_id‘ column with the column containing close to only 50k unique values. We felt like this repetition is non-trivial since people who have already been readmitted are more susceptible to readmittance. We therefore created a new column indicative of how many times a particular patient has had encounters nicknamed ‘pat_cnt‘. We computed this simply by using the number of times a particular patient id occurred in the dataset. This was a breakthrough and led to a solid 5% increase in performance. If we include the number of occurrences

in both train and test splits of the data, the score improved by a massive 9%. We therefore concluded that this is the single most important feature of our dataset.

- **Reducing category complexity**: Another method we sought to decrease the data's complexity was to decrease the number of categories in categorical columns. While we partially addressed it while grouping the similar ICD9 codes in 'diag' columns, doing it for the other columns was not so simple. We employed the following two methods to achieve this:
  - **Manually grouping :** We manually grouped related categories into the same category. In some cases, this was trivial as some categories had synonymous words representing them. In other cases, it was a lot more difficult as grouping needed domain knowledge. However, in all our experiments trying different combinations, the final score always decreased on reducing number of columns.
  - **Using clustering :** We attempted to use unsupervised learning to group related categories. For each category in a categorical column, we computed the proportion of the three classes of 'readmission_id'. We then applied K-means clustering on this 3-dimensional space to compute the new categories. This also did not lead to an improvement in the score but only a decrease.

This led us to believe that there might be some information even in the redundant categories.

I. *Notable Anomalies*

Decisions not aligning with best practices but improving the score included using test data for 'pat_cnt' computation and using 'enc_id' and 'patient_id' for training.

II. Non-neural methods

A. *Methodology*

1) ***Models Used:***

1. *Bagging Models:*
   - RandomForestClassifier
   - ExtraTreesClassifier
2. *Boosting Models:*
   - XGBoostClassifier
   - LightGBMClassifier
   - LGBM dart variant
   - CatBoostClassifier
   - HistGradientBoostClassifier
3. *Voting:*
   - VotingClassifier

2) ***Evaluation Process:***

1. *Cross-Validation:*
   - Performance of each model was assessed using cross-validation scores.
2. *Kaggle Leaderboard:*
   - Model performance was compared on the Kaggle leaderboard.
3. *Voting Classifier:*
   - A VotingClassifier was trained using the softmax of Kaggle results as weights.

B. *Results*

Model	**Kaggle Leaderboard Score (%)**
RandomForestClassifier (rf)	72.7
ExtraTreesClassifier (et)	71.5
XGBoostClassifier (xgb)	73.2
LightGBMClassifier (lgb)	73.5
LGBM dart variant (dart)	73.1
CatBoostClassifier (cb)	73.0
HistGradientBoostClassifier (hgb)	73.0
VotingClassifier (voting)	73.5

TABLE III
KAGGLE LEADERBOARD RESULTS FOR EACH MODEL.

C. *Code Overview*

1. *Data Loading and Preprocessing:*
   - Data is loaded and preprocessed using methods from the `preprocessing.py` file.
2. *Cross-Validation Function:*
   - A function (`cross_val_score`) is defined to perform cross-validation for a given estimator.
3. *Model Training:*
   - The listed ensemble models are trained using the cross-validation function, and their scores are recorded.
4. *Submission Generation:*
   - Predictions from the trained models are generated, and submission files are created for each model.
5. *Voting Classifier:*
   - Softmax of Kaggle results is used as weights for training a VotingClassifier.

D. *Conclusion*

The ensemble models, particularly the VotingClassifier, demonstrated competitive performance on the Kaggle leaderboard. The combination of diverse models through ensembling enhanced predictive accuracy. The results showcase the effectiveness of ensemble methods in addressing the Canadian Hospital Re-admittance Challenge.

III. SVM

A. *Pre-processing*

The same pre-processing steps have been used as detailed in Section I. It is to be noted that SVM classifiers were performing very poorly (getting validation accuracy in the ballpark of 53%) before the data was normalized using a StandardScaler. The reason for this performance jump (also observed in Section IV) could be that SVM is sensitive to feature scales. Normalizing the data ensures all features contribute equally to the decision boundary, preventing bias toward features with larger scales and enhancing the model's ability to capture patterns in the data, resulting in higher accuracy.

B. Kernel selection

We tried out two different kernels :

- **Polynomial**
  To obtain the degree of polynomial kernel to be used, we ran experiments with 5 values. The results are summarised in Fig. 1.

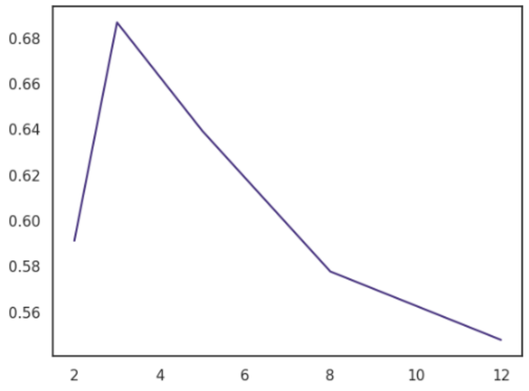
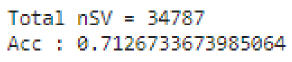


Fig. 1. Validation accuracy scores of SVCs with polynomial kernels - accuracy score on the Y-axis and degrees on the X-axis

  Clearly, the most suitable polynomial kernel was the one with degree 3.
- **Radial Basis Function**
  In this case the work was cut out for us as there are no additional hyper-parameters to tune. Using this kernel, we could attain a validation accuracy of 71.27% (Fig 2)



```
Total nSV = 34787
Acc : 0.7126733673985064
```

Fig. 2. Validation accuracy using RBF kernel

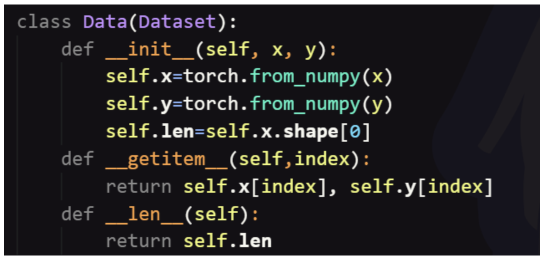
C. Final results

Upon submitting with the predictions of the SVC with kernel = 'rbf', we attained a score equal to **0.715**.

IV. Neural net

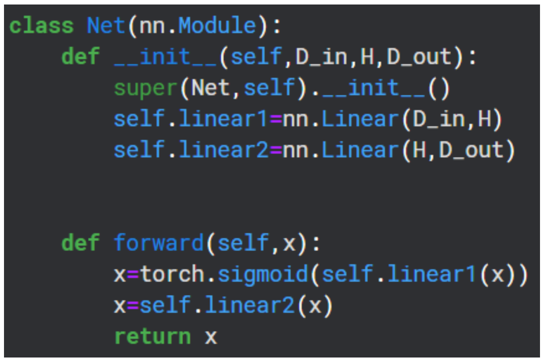
In this section, we'll mainly list down our training approach with neural network, difficulties we faced and the solution we obtained.

We used PyTorch library for building our neural network model and our model achieves a score of 72% on kaggle.



```
class Data(Dataset):
    def __init__(self, x, y):
        self.x=torch.from_numpy(x)
        self.y=torch.from_numpy(y)
        self.len=self.x.shape[0]
    def __getitem__(self,index):
        return self.x[index], self.y[index]
    def __len__(self):
        return self.len
```

Fig. 3. Dataset class



```
class Net(nn.Module):
    def __init__(self,D_in,H,D_out):
        super(Net,self).__init__()
        self.linear1=nn.Linear(D_in,H)
        self.linear2=nn.Linear(H,D_out)


    def forward(self,x):
        x=torch.sigmoid(self.linear1(x))
        x=self.linear2(x)
        return x
```

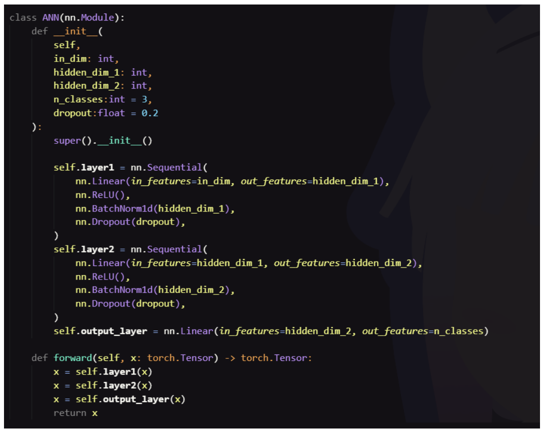
Fig. 4. Simple Neural Netowrk for baseline figures

A. *Setting up the code :*

We used the same pre-processed data obtained after EDA. We needed our data to be compatible with pytorch tensors so we created a Data class (5) which is a wrapper to the original Dataset class. Data class takes in the data and converts it to pytorch tensors. We built out 2 models, one using the github repository of ML course tutorial as reference ([1]) and the other, a very simple model written from scratch for baseline figures.

B. *Architecture of the Neural Network*

*1) **Baseline Model:*** Baseline model has three layers, input layer, one hidden layer and an output layer. In our experiments,



```
class ANN(nn.Module):
    def __init__(
        self,
        in_dim: int,
        hidden_dim_1: int,
        hidden_dim_2: int,
        n_classes:int = 3,
        dropout:float = 0.2
    ):
        super().__init__()

        self.layer1 = nn.Sequential(
            nn.Linear(in_features=in_dim, out_features=hidden_dim_1),
            nn.ReLU(),
            nn.BatchNorm1d(hidden_dim_1),
            nn.Dropout(dropout),
        )
        self.layer2 = nn.Sequential(
            nn.Linear(in_features=hidden_dim_1, out_features=hidden_dim_2),
            nn.ReLU(),
            nn.BatchNorm1d(hidden_dim_2),
            nn.Dropout(dropout),
        )
        self.output_layer = nn.Linear(in_features=hidden_dim_2, out_features=n_classes)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.output_layer(x)
        return x
```

Fig. 5. Final Model

input layer had neurons equal to the number of features in the dataset and we chose the hidden layer to have 100 neurons and output layer had 3 neuron as that is the number of classes we are trying to predict. We chose the activation function to be sigmoid.

*2) **Final Model**:* Based on [[1]], we chose our final model to have 4 layers, one input layer having neurons equal to the number of features in the dataset, two hidden layer with 100 and 20 neurons respectively and one output layer with 3 neurons which is equal to the number of classes. Further, the model had ReLU as the activation function and 0.2 with 0.2 dropout rate to prevent overfitting.

*C. **Experiments :***

We first trained our baseline model and then the final model on the preprocessed data and found that both model were achieving only 53% accuracy on train and validation data (the model was only predicting class 2). Upon further inspection, we found that the loss was very unstable as shown in figure 6. There could have been multiple reason for this :

- *Activation Function :* Sigmoid easily saturates (small region where changes occur, most of the values are almost 0 or 1). ReLU are not prone to saturation [2].
- It can happen when data is not normalized. Normalization deals with outliers, deviation and anomalies. If we do not account for these then the calculation can become disproportionate - an unequal weight assigned to some parts of the system that should not have that much of an influence. [3]
- *Overfitting :* This can also happen when the network is overfitting the data. To prevent overfitting, we can try batch normalization with high learning rate, dropout, smaller batch size or early stopping. [2]

For fixing our unstable loss vs steps, we found that normalization was required. After standard scaling the data using the standard scaler from scikit-learn library, we found improvement in loss vs steps as shown in figure 7 and found improvement in accuracy upto 59%. However, model was still not predicting class 0.

Next we tried hyperparameter tuning and with one hot encoding of data instead of ordinal encoding and achieved 72% score on kaggle with training loss vs epochs as shown in figure 8

*D. **Training time :***

We trained our model on 2 cpu and 3 gpu to compare the training time when batch size selected was 64.

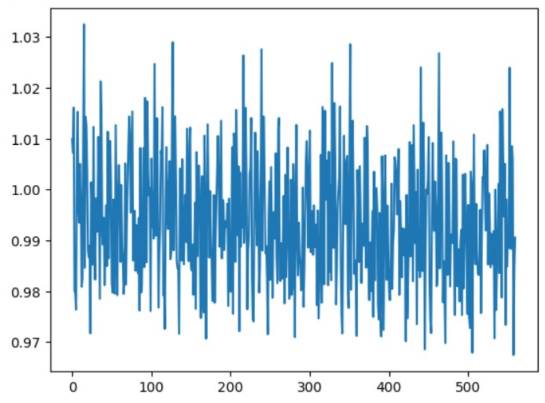


Fig. 6. Plot of loss vs steps

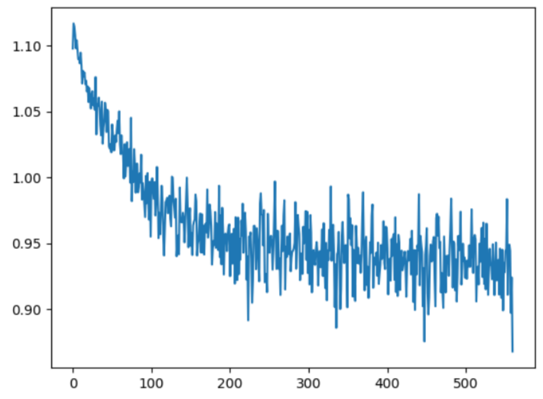


Fig. 7. Plot of loss vs steps (After Improvement)

Processor	Core Count	Time (per epoch)
Ryzen 8 4800H	8 vCPU cores	2.7s
Intel Xeon 2.20 GHz CPU	4 vCPU cores	2.8s
NVIDIA GeForce GTX 1650	896 CUDA Cores	2.5s
NVIDIA T4 x2 GPU	2560 Cuda Cores	2.06s
NVIDIA Tesla P100 GPU	3584 Cuda cores	1.97s

We can see the difference between time in training on gpu and cpu is not much (at max 1s). The training time was mainly affected by the batch size that we selected as it allows for parallelism. Below we list the training time taken for different selection of batch sizes when training on P100 GPU :

Batch Size	Time (per epoch)
2	48s
4	23s
8	12s
16	6s
64	1.97s

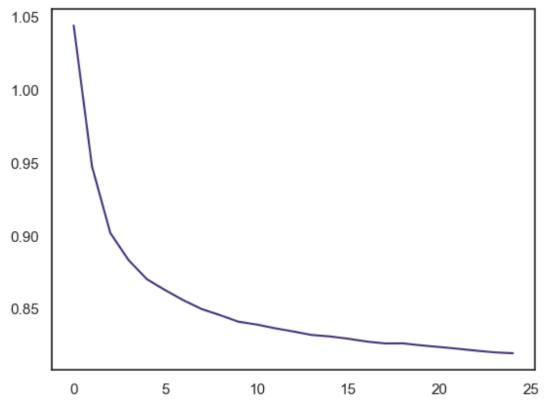


Fig. 8. Final plot of loss vs epochs

*E. **Final Result :***

With one input layer consisting of 167 neurons, two hidden layer with 100 and 20 neuron respectively and one output layer with 3 neurons and with batch normalization with dropout rate of 0.2 and the data standard scaled, we were able to achieve a score of 71% on validation data and 72% on Kaggle.

REFERENCES

[1] https://github.com/sarthakharne/Machine-Learning-TA-Material-Fall-2023
[2] https://stackoverflow.com/questions/55894132/how-to-correct-unstable-loss-and-accuracy-during-training-binary-classificatio
[3] https://www.quora.com/What-happens-if-I-do-not-normalise-the-training-data-for-a-neural-network