HOME CREDIT DEFAULT RISK

Team Members

Sagar Prabhu
Tanmay Mahindrakar
Bhakti Patrawala
Sohan Kakatkar

1 Phase Leader Plan:

Phases	Leader
Phase 1	Bhakti Patrawala
Phase 2	Sagar Prabhu
Phase 3	Tanmay Mahindrakar
Phase 4	Sohan Kakatkar

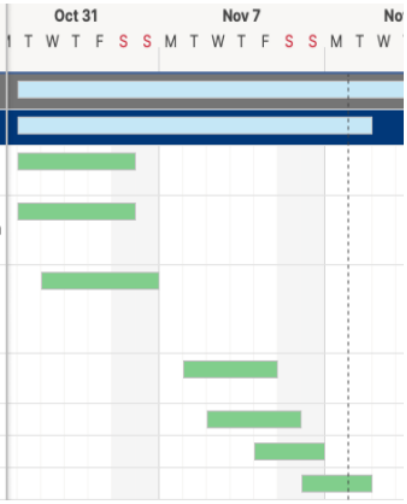
Phase	Description	Contributor(s)
Phase 1: Project Proposal	Write Abstract, Analyze data from dataset.	Tanmay Mahindrakar
Phase 1: Project Proposal	Data Description, Machine Learning Algorithms description and its functional exploration in a dataset with its cost, loss functions.	Sagar Prabhu
Phase 1: Project Proposal	Description of other metrics in Machine learning algorithms, usage of Machine Learning pipelines.	Bhakti Patrawala
Phase 1: Project Proposal	Creation and description of pipeline diagram.	Sohan Kakatkar
Phase 1: Project Proposal	Gantt Chart Diagram	All
Phase 2: EDA + Baseline Pipeline Development	EDA and preprocessing for 4 tables and define the metrics used.	Sohan Kakatkar
Phase 2: EDA + Baseline Pipeline Development	EDA and preprocessing for 3 tables and define the metrics used.	Sagar Prabhu
Phase 2: EDA + Baseline Pipeline Development	Design and build baseline pipelines with code.	Tanmay Mahindraka
Phase 2: EDA + Baseline Pipeline Development	Describe and build baseline pipelines with Block Diagram and table of experimental results.	Bhakti Patrawala
Phase 2: EDA + Baseline Pipeline Development	Brief Report related to EDA and preprocessing, abstract.	Sagar Prabhu, Sohan Kakatkar
Phase 2: EDA + Baseline Pipeline Development	Brief Report related to baseline pipelines and experiments.	Bhakti Patrawala, Tanmay Mahindrakar
Phase 2: EDA + Baseline Pipeline Development	Video Presentation	All

Phase 3: Feature Engineering/ Hyper-parameter tuning	Feature Engineering and Feature Selection	Sohan Kakatkar
Phase 3:Feature Engineering/ Hyper-parameter tuning	Block Diagram of pipelines, Model Development using Linear, Logistic, SVM	Sagar Prabhu
Phase 3: Feature Engineering/ Hyper-parameter tuning	Model Development using Decision Tree Algorithm, Random Forest Algorithm	Bhakti Patrawala
Phase 3: Feature Engineering/ Hyper-parameter tuning	Model Validation	Tanmay Mahindrakar
Phase 3: Feature Engineering/ Hyper-parameter tuning	Hyperparameter Tuning	Bhakti Patrawala
Phase 3: Feature Engineering/ Hyper-parameter tuning	Presentation	All
Phase 4: Final Submission	Model Pipelines	Bhakti Patrawala, SagarPrabhu
Phase 4: Final Submission	Hyperparameter Tuning	Sohan Kakatkar
Phase 4: Final Submission	Graphical Representation of graphs performance	Sagar Prabhu
Phase 4: Final Submission	Validation	Tanmay Mahindrakar
Phase 4: Final Submission	Write abstract, metrics analysis, loss function and Summary	Sohan Kakatkar
Phase 4: Final Submission	Final Report	All
Phase 4: Final Submission	Slides and Video presentation	All

2 Gantt Chart

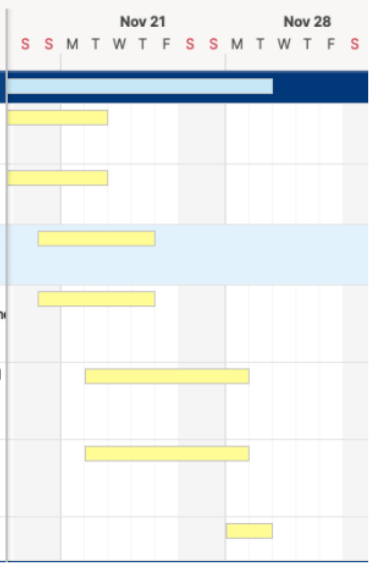
Phase 1:

Task Name	Status	H...	Start Date	End Date	Assigned To	Description
Project			11/01/22	12/15/22		
Phase 1			11/01/22	11/15/22		
Project Proposal	Complete		11/01/22	11/05/22	Tanmay Mahindrakar	Abstract, Analyze data from dataset.
Project Proposal	Complete		11/01/22	11/05/22	Sagar Prabhu	Data Description, Machine Learning Algorithms description and its functional exploration
Project Proposal	Complete		11/02/22	11/06/22	Bhakti Patrawala	Description of other metrics in Machine learning algorithms, usage of Machine Learning pipelines.
Project Proposal	Complete		11/08/22	11/11/22	Sohan Kakatkar	Creation and description of pipeline diagram.
Project Proposal	Complete		11/09/22	11/12/22	Sagar, Tanmay	Phase Leader Plan
Project Proposal	Complete		11/11/22	11/13/22	Sohan, Bhakti	Credit Assignment Plan
Project Proposal	Complete		11/13/22	11/15/22	All	Gantt Chart Diagram



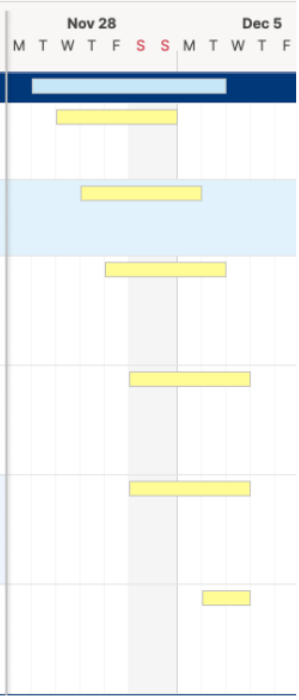
Phase 2:

Task Name	Status	H...	Start Date	End Date	Assigned To	Description
Phase 2			11/16/22	11/29/22		
EDA + Baseline Pipeline Development	In Progres		11/16/22	11/22/22	Sohan Kakatkar	EDA and preprocessing for 4 tables and define the metrics used.
EDA + Baseline Pipeline Development	In Progres		11/16/22	11/22/22	Sagar Prabhu	EDA and preprocessing for 3 tables and define the metrics used.
EDA + Baseline Pipeline Development	In Progres		11/20/22	11/24/22	Tanmay Mahindrakar	Design and build baseline pipelines with code.
EDA + Baseline Pipeline Development	In Progress		11/20/22	11/24/22	Bhakti Patrawala	Describe and build baseline pipelines with Block Diagram an table of experimental results.
EDA + Baseline Pipeline Development	In Progress		11/22/22	11/28/22	Sagar, Sohan	Brief Report related to EDA and preprocessing, abstract.
EDA + Baseline Pipeline Development	In Progress		11/22/22	11/28/22	Bhakti, Tanmay	Brief Report related to baseline pipelines and experiments.
Video Presentation	In Progress		11/28/22	11/29/22	All	Video Presentation



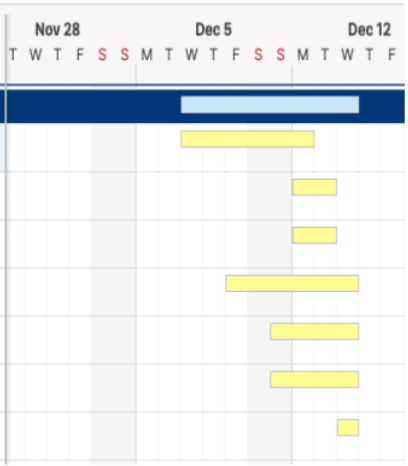
Phase 3:

Task Name	Status	H...	Start Date	End Date	Assigned To	Description
Phase 3			**11/29/22**	**12/06/22**		
Phase 3: Feature Engineering/ Hyper-parameter tuning	In Progres		11/30/22	12/04/22	Sohan Kakatkar	Feature Engineering and Hyperparameter tuning in pipeline forms
Phase 3: Feature Engineering/ Hyper-parameter tuning	In Progres		12/01/22	12/05/22	Sagar Prabhu	Block Diagram of pipelines, Model Development using Linear, Logistic, SVM
Phase 3: Feature Engineering/ Hyper-parameter tuning	In Progres		12/02/22	12/06/22	Bhakti Patrawala	Model Development using Decision Tree Algorithm, Random Forest Algorithm
Phase 3: Feature Engineering/ Hyper-parameter tuning	In Progres		12/03/22	12/07/22	Tanmay Mahindrakar	Model Validation
Phase 3: Feature Engineering/ Hyper-parameter tuning	In Progres		12/03/22	12/07/22	Bhakti Patrawala	Hyperparameter Tuning
Phase 3: Feature Engineering/ Hyper-parameter tuning	In Progress		12/06/22	12/07/22	All	Presentation



Phase 4:

Task Name	Status	H...	Start Date	End Date	Assigned To	Description
Phase 4			**12/07/22**	**12/14/22**		
Phase 4: Final Submission	In Progres		12/07/22	12/12/22	Bhakti Patrawala	Model Pipelines
a file to this row : Final Submission	In Progres		12/12/22	12/13/22	Sohan Kakatkar	Hyperparameter Tuning
Phase 4: Final Submission	In Progress		12/12/22	12/13/22	Sagar Prabhu	Graphical Representation of graphs performance
Phase 4: Final Submission	In Progress		12/09/22	12/14/22	Tanmay Mahindrakar	Validation
Phase 4: Final Submission	In Progress		12/11/22	12/14/22	Sohan Kakatkar	Write abstract, metrics analysis, loss function and Summary
Phase 4: Final Submission	In Progress		12/11/22	12/14/22	All	Final Report
Phase 4: Final Submission	In Progress		12/14/22	12/14/22	All	Slides and Video presentation



3 Credit Assignment Plan

Task	Phase	Assigned to	Days	Start	End
Phase leader and credit assignment plan	1	Team	1	11/01/2022	11/01/2022
Project proposal abstract	1	Tanmay Mahindrakar	1	11/03/2022	11/03/2022
Data Understanding	1	Sohan Kakatkar	1	11/05/2022	11/05/2022
ML algorithm exploration and analysis	1	Sagar Prabhu	2	11/05/2022	11/06/2022
Algorithm metric evaluation	1	Bhakti Patrawala	2	11/07/2022	11/08/2022
ML pipelines	1	Sohan Kakatkar	2	11/12/2022	11/12/2022
Team photo	1	Team	1	11/13/2022	11/13/2022
Proposal post submission	1	Team	1	11/15/2022	11/15/2022
Download data and perform EDA	2	Sohan Kakatkar	2	11/16/2022	11/17/2022
Missing values analysis	2	Sohan Kakatkar	2	11/19/2022	11/20/2022
Data cleaning & processing	2	Sagar Prabhu	2	11/17/2022	11/18/2022
Feature Engineering	2	Tanmay Mahindrakar	2	11/18/2022	11/19/2022
Design and construct baseline pipeline	2	Tanmay Mahindrakar	3	11/22/2022	11/24/2022
Result table	2	Bhakti Patrawala	2	11/18/2022	11/19/2022
Pipeline block diagram	2	Sagar Prabhu	2	11/21/2022	11/22/2022
Documentation(report, conclusion)	2	Bhakti Patrawala	1	11/23/2022	11/23/2022
Submit Video with all team members	2	Team	1	11/29/2022	11/29/2022
Detailed feature engineering if required	3	Sohan Kakatkar	2	11/30/2022	12/01/20221 2/01/2022
Kaggle competition submission	3	Team	2	12/04/2022 12/04/2022	12/05/20221 2/05/2022
Report, conclusion and plan for next phase	3	Tanmay Mahindrakar	1	11/30/20221 1/30/2022	11/30/202211 /30/2022
Video presentation	3	Team	1	12/03/2022 12/03/2022	12/03/20221 2/03/2022

Proposal post submission	3	Team	22	12/05/2022 12/05/2022	12/06/20221 2/06/2022
Hyperparameter Tuning	4	Sagar Prabhu	2	12/07/2022 0712/	12/08/2022
Validation	4	Bhakti Patrawala	2	12/08/2022	12/09/2022
Graphical Representation of graphs performance	4	Sohan Kakatkar	2	12/08/2022	12/09/2022
Write abstract, metrics analysis, loss function and Summary	4	Tanmay Mahindrakar	1	12/11/2022	12/11/2022
Final report	4	Team	1	12/12/2022	12/12/2022
Slides and Video presentation	4	Team	1	12/14/2022	12/14/2022

Abstract

The dataset contains a total of 7 tables such as application, bureau balance, credit card balance installments, previous applications providing valuable data for individuals. The phase is divided into EDA, Visual EDA, Feature Selection and Engineering, Modeling Pipelines with one-hot encoding, Hyperparameter tuning with decision tree ensemble methods using PyTorch and lastly results with conclusion. We were tackling Feature Engineering to select the most significant features utilizing feature selection, handling the missing data, data normalization and addition of essential features. In this process of normalization, we brought more significant features such as DAYS-CREDIT, AMT-CREDIT and EXT-SOURCES into picture. During pipelining, we had tested a series of binary classification models in which Decision Tree Classifier outperformed the other models. Additionally,we used Grid search from sklearn with pipeline and cross validation to perform hyperparameter tuning using Random Forest, Decision Tree, AdaBoost, LGB, Multilayer Perceptron and xgBoost. Additionally, several metrics are utilized like Root Mean Square Error, Mean Square Error, Log loss, MAPE and Accuracy score.

In Pytorch Classification, we used the Sequential model to train our dataset. From the following Architectures, we experimented many combinations of architecture like Softmax and ReLu with optimizers changing from Adam to SGD. We also implemented loss function of CXE, MSE and MSE + CSE by altering the tensor dimensions in our classification model. We also implemented Regression Pytorch model using Softmax model with MSE scores. We achieved best test accuracy of 0.92 for Decision Tree algorithm. Further, we obtained kaggle public and private scores as 0.726 and 0.716 respectively. For getting the kaggle scores, we computed metrics like Mean squared error, found accuracy and precision scale for which we understood how to go around. Further, we analyzed Pytorch curves which were plotted for Classification and Regression MSE, CXE. Over there, we extracted insights with the approach of dropping some features which had a uniform or lower valued curve as compared to other features.

4 Introduction

Home Credit Default Risk deals with classifying loan applicants based on variety of client data including previous credit, previous applications, repayment history for the previously disbursed credits, etc. to predict their repayment abilities for a positive loan experience. This project aims to provide safe borrowing experience to for the client by ensuring that clients that are capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.

4.1 Project Description

In this project, considering our goal to predict whether or not a client will repay a loan using Classification Machine Learning Algorithms, we will firstly require implementation of data handling and cleaning techniques. EDA is utilized to perform interactive and hypothesis-driven data exploration with the intention of removing outliers, filling the missing values or dropping insignificant columns with respect to the Target column. Than we are performing correlation analysis followed by Pipeline modelling on series of Machine Learning Classification algorithms. Here, the goal is to get optimum values for accuracy on the test and validation set and to prevent over-fitting on the training data.

4.1.0.1 We will be using the TensorBoard to keep track of the training progress of the PyTorch model.We are also planning on developing and implementing multitask loss function in PyTorch.

5 Workflow diagrams

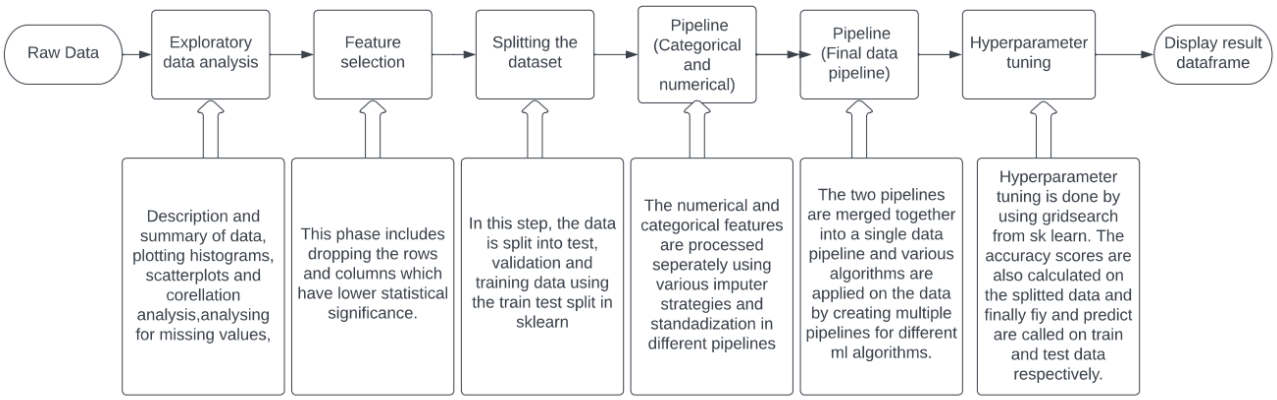


Figure 1: Model pipeline Architecture

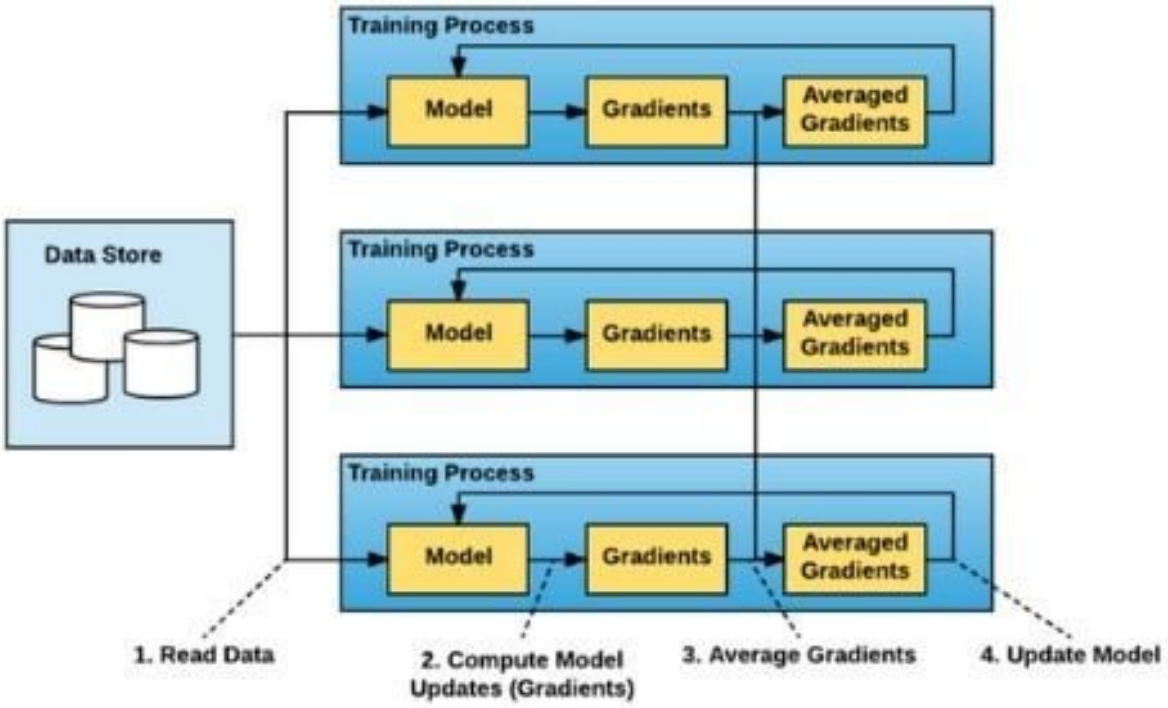


Figure 2: PyTorch Work Flow

6 Data Description

Here, we are given a set of 7 sets of data which can be used for our target classification. We can observe some relational structure in the sets which means we need to be vigilant while dropping any row or column considering its foreign and primary keys. The main train set is named as application-test which has 121 columns with the target attribute. Previous dated data can be about the loan which had been bought from other institutions which were noted by the Credit Bureau Union can be seen in bureau set and bureau-balance set which are both relational by nature of their data. We are also given the monthly balance of the users which is considered a paramount set named as pos-cash-balance set. Another set which consists the data of the user at the time of previous applications is monitored in previous-application set. There are 2 more sets which give behavioral data of past credit card balance (monthly) and information related to past loans in credit-card-balance set and instalments-payments set respectively. Here, we can see we have both numerical and categorical values which suggests the need of one-hot-encoding to create a singular pipeline. Also, we can see also some Nan values which need to be handled efficiently so that they don't interfere in our model.

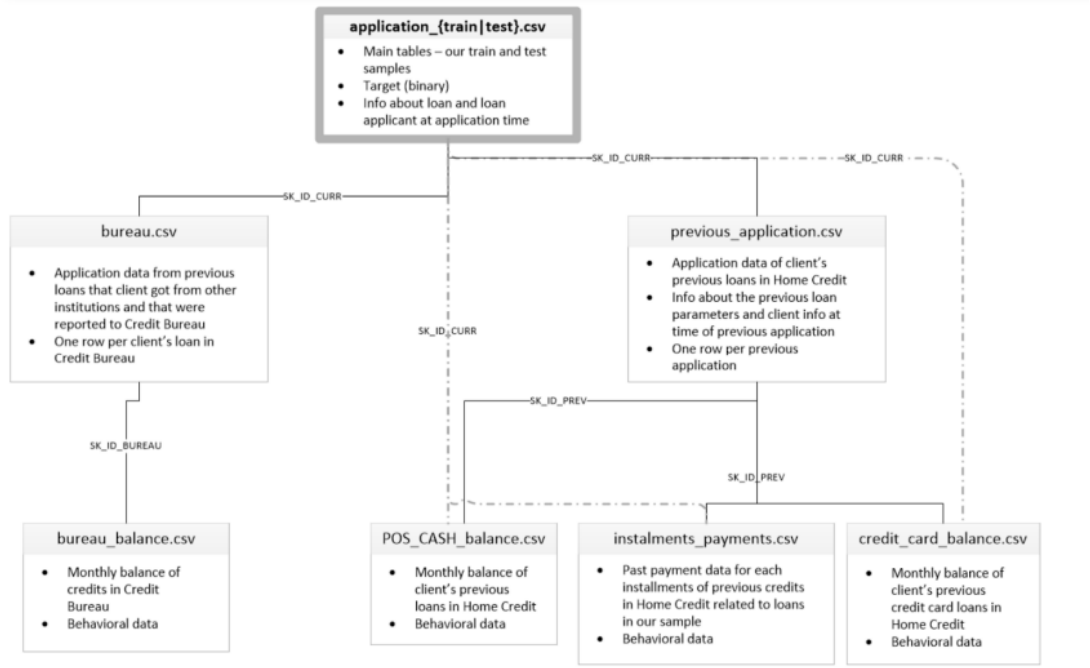


Figure 3: Model Pipeline

7 Exploratory Data Analysis (EDA)

Exploratory Data Analysis is used for analyzing data sets with detecting and removing outliers, analyzing missing values by dropping tables with N/A values or filling the missing column with imputed values. Utilizing EDA assisted us to understand the dataset variables and the relationship among them. List of Built-in functions utilized for each dataset are: 1. info(dataset): Find number of entries, datatypes and memory usage of dataset 2.shape(dataset): To find the shape of the dataset 3.head(dataset): To print the first n entries to understand the kind of data we are dealing with. 4. tail(dataset):To print the last n entries to understand the kind of data we are dealing with. 5. describe(dataset): To describe the dataset i.e. find the metrics like mean, median, min and max values, quartile, etc. of the dataset. 6. We have also calculated numerical, categorical and all features of a dataset 7. Count of each data type present in dataset 8. The value counts in dataset

- Missing Value Analysis and Plots to visualize the missing values:

Here we have figured out the presence of columns with missing values and the solutions to it is in the form of dropping insignificant columns or filling the missing values with uniform value. We have also plotted the respective significant missing plot for each datasets creating a function for the same. The Missing plot and the function for application train dataset:

```
def Missing_Plot(dataset):
    plt.figure(figsize=(210,50))
    sns.displot(
    data=datasets[dataset].iloc[: ,20 :60].isna().melt(value_name="missing"),
    y="variable",
    hue="missing",
    multiple="fill",
    aspect=3
    ).set(title='Missing Values Plot')
```

```
Missing_Plot("application_test")
```

```
<Figure size 15120x3600 with 0 Axes>
```

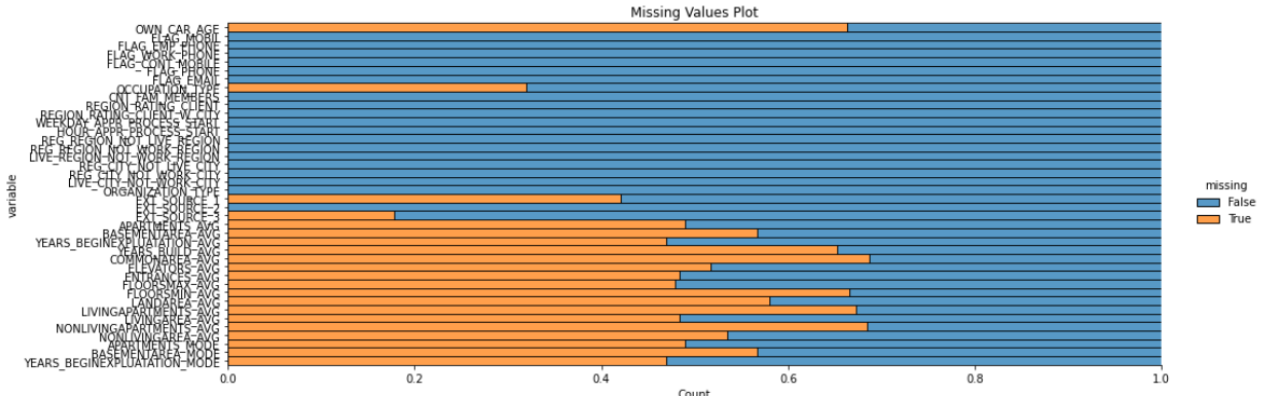


Figure 4: Missing Analysis Plot

The Missing plot and the function for Bureau dataset:

```
Missing_Plot1("bureau")
```

```
<Figure size 15120x3600 with 0 Axes>
```

Figure 5: Missing Analysis Plot

- Correlation Analysis:

Correlation Plot with the Target Variable:

```
03]: plt.figure(figsize = (70,70))
     corrMap = sns.heatmap(datasets["application_train"].corr(), vmin=-1, vmax = 1, annot=True)
```

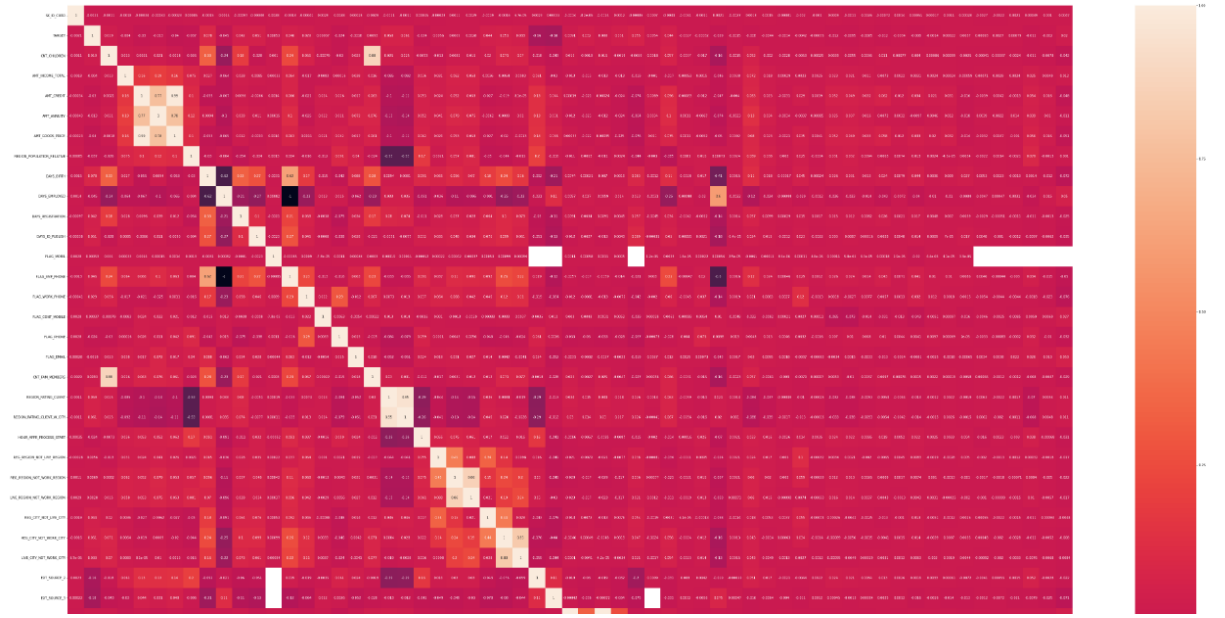


Figure 6: Correlation Plot with the Target Variable

Correlation map of highly positive correlated features of Application train dataset to Target:

```
: # Correlation map of highly positive correlated features of application train to TARGET
  plt.figure(figsize = (50,50))
  corr_cols = ['DAYS_BIRTH', 'REGION_RATING_CLIENT_W_CITY','REGION_RATING_CLIENT','DAYS_LAST_PHONE_CHANGE', 'DAYS_ID_PUBLISH',
               'REG_CITY_NOT_WORK_CITY','FLAG_EMP_PHONE','REG_CITY_NOT_LIVE_CITY', 'FLAG_DOCUMENT_3', 'TARGET']
  corrMap = sns.heatmap(datasets["application_train"][corr_cols].corr(), vmin=-1, vmax=1, annot=True)
```

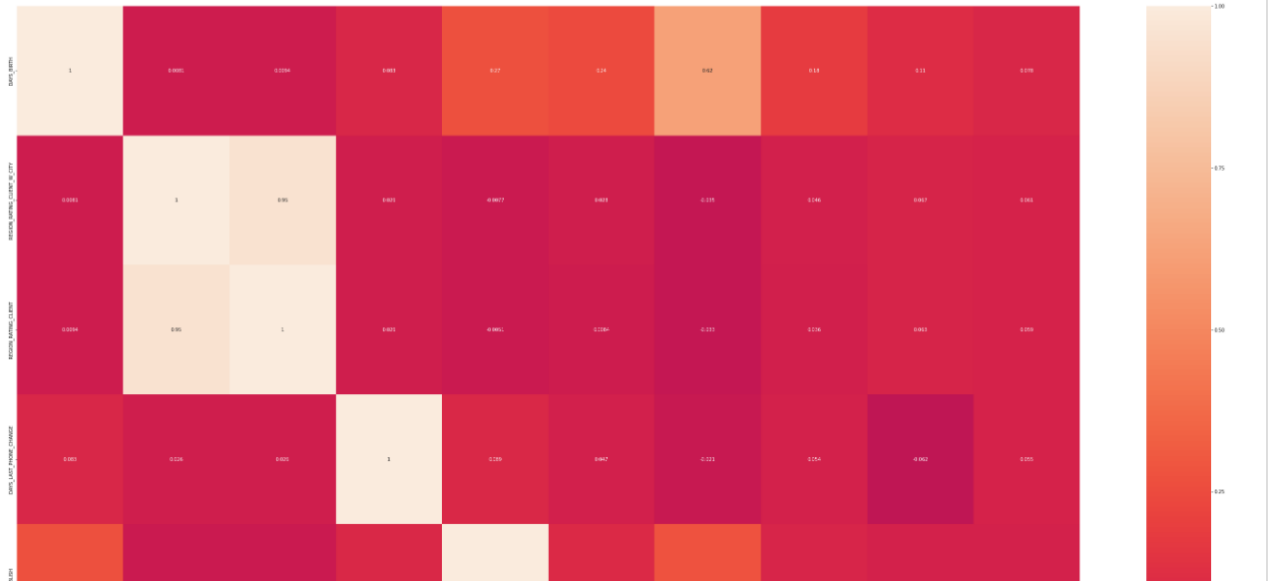


Figure 7: Correlation map of highly positive correlated features

Below is the histogram to correlated values in application train dataset with respect to given column

```
plt.hist((datasets["application_train"]['DAYS_BIRTH']),edgecolor='red', color='royalblue', bins =30)
plt.title('Days Birth Data'); plt.xlabel('Correlated values'); plt.ylabel('Count');
```

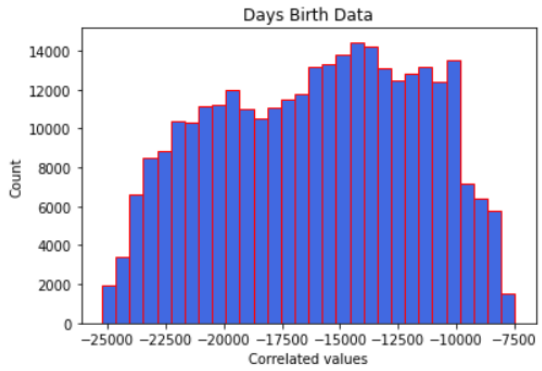


Figure 8: Correlation Analysis

8.0.1 People are dealing more with Cash loans

```
]: categorical_plot('NAME_CONTRACT_TYPE', 'Spectral')
```

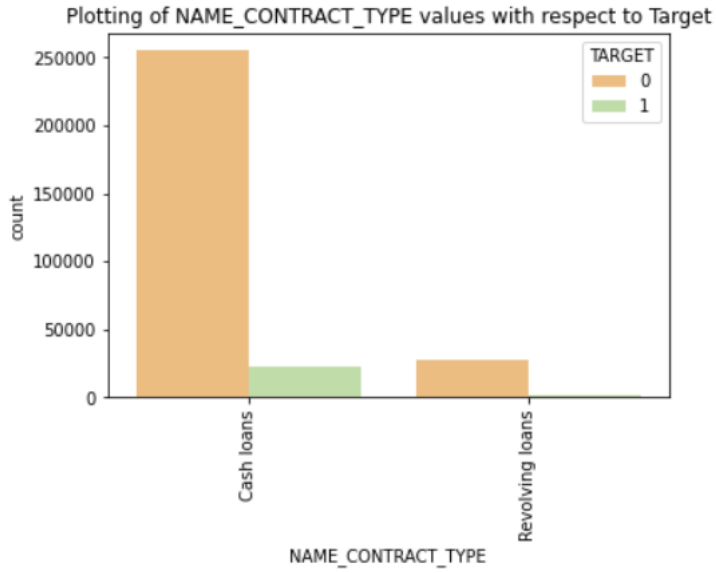


Figure 9: Correlation Analysis

```
]: categorical_plot('NAME_INCOME_TYPE', 'coolwarm')
```

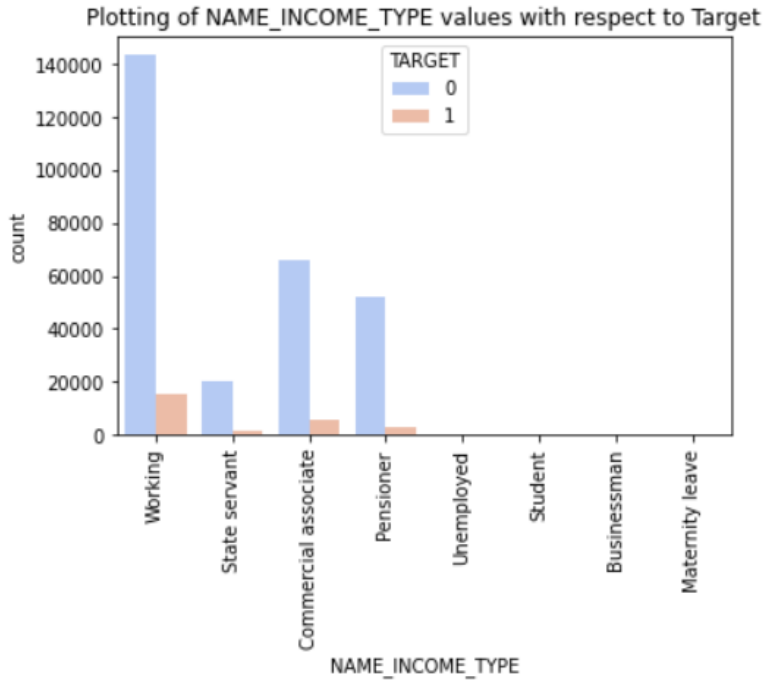


Figure 10: Correlation Analysis

8 Feature Engineering

Here, we are using Feature Engineering so that we can select the most significant and essential features which would be transformed into refined and meaningful features that would suit the need of the model.

- Feature Engineering utilizes various data engineering techniques like feature selection of relevant features, handling the missing data, data normalization and addition of essential features which can best train or model.
- We have focused more on Feature Engineering as it is important to engineer inputted data to model to extract significant features.
- The categorical feature is one hot encoded so that they can be used appropriately by the machine learning algorithms.

8.1 Additional Features added to Training Data:

- Computed Average values for all features in Previous application data table and merged this feature column in application training dataset.
- Computed Average values for all features in Installments Payment data table and we have merged this feature column with the application training dataset.

8.2 Impact of the new features added to Training Data:

- Here, we utilized the added featured application training data table to our lightgbm.LGBMClassifier to train the model.
- Includes test and train split validation and model runs until validation scores doesn't improve for 150 rounds and provides the best iteration AUC score.
- Feature Importance: Plotted the feature importance of lgb model using the added feature with max numerical features.

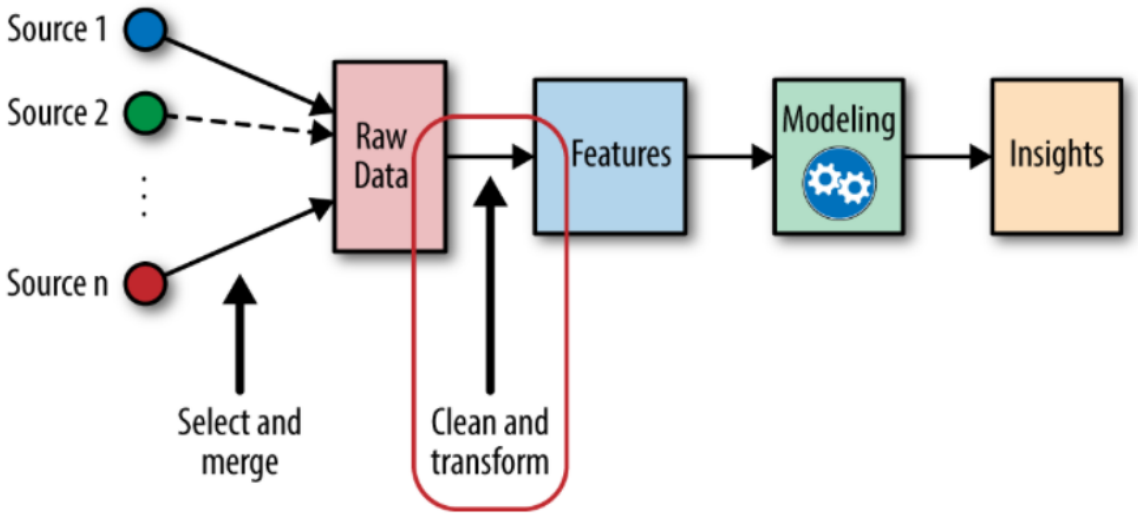


Figure 11: Feature Engineering Steps

8.3 Why we chose the method:

- We are using this method so that we can select the most significant and essential features which would be transformed into refined and meaningful features that would suit the need of the model.
- Feature Importance: Feature importance provides a deeper insights of the features significance and exploits the level of understanding on which feature to use and to be added in the model

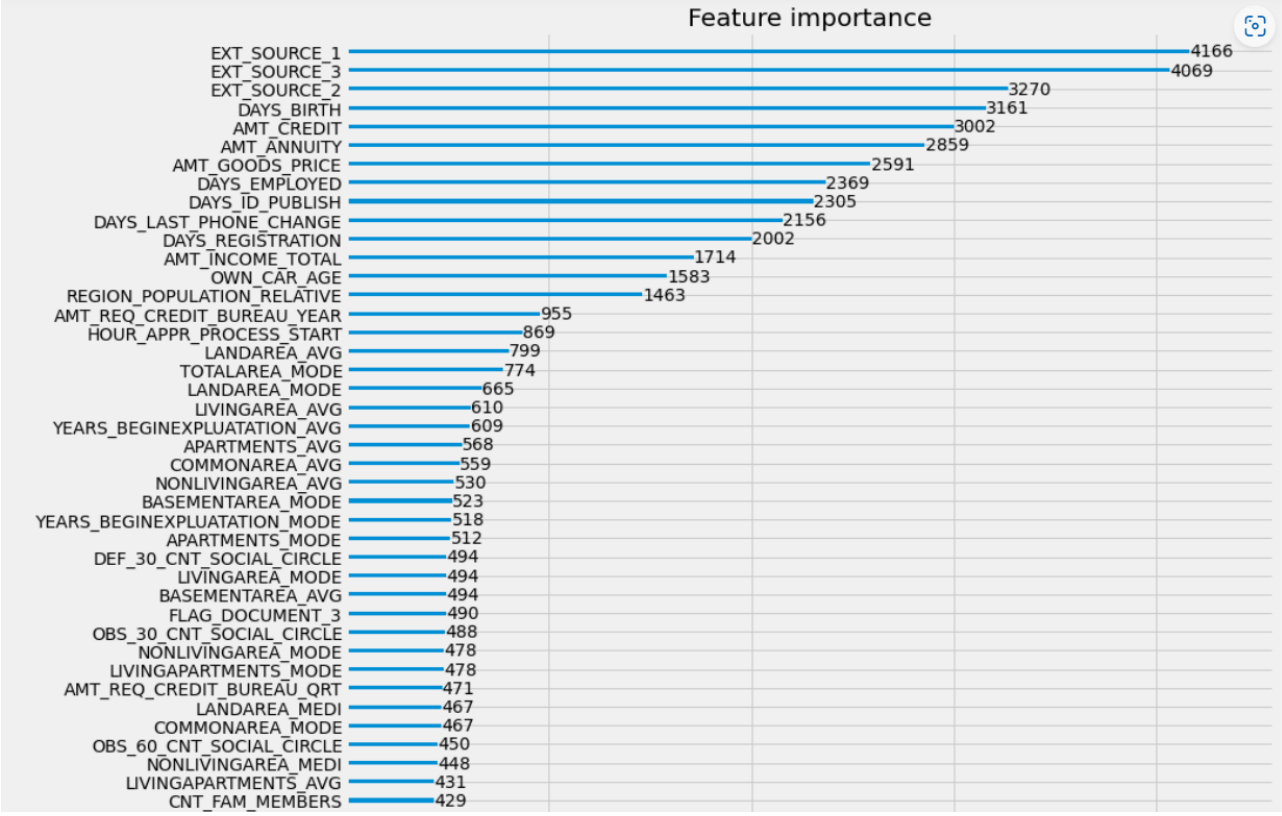


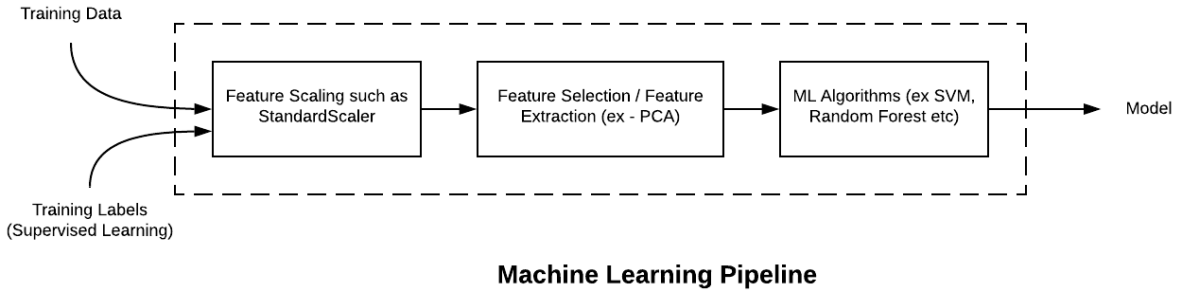
Figure 12: Feature Importance

9 Modelling Pipeline

The given dataset contains the following obstacles in smooth execution of machine learning algorithms -

- There are many missing values in the dataset.
- There numerical and categorical and numerical values need to handled differently
- The numerical data needs to be standardized since there is huge difference in the range of various features
- The categorical feature needs to be one hot encoded so that they can be used appropriately by the machine learning algorithms.

The missing values are handled by doing exploratory data analysis. In order to systematically handle the categorical and numerical features, modelling pipeline play a very significant role. Machine Learning (ML) pipeline, theoretically, represents different steps including data transformation and prediction through which data passes. The outcome of the pipeline is the trained model which can be used for making the predictions. Sklearn.pipeline is a Python implementation of ML pipeline. Instead of going through the model fitting and data transformation steps for the training and test datasets separately, you can use Sklearn.pipeline to automate these steps. Here is a diagram representing a pipeline for training a machine learning model based on supervised learning.



Machine Learning Pipeline

9.1 Families, number of input features and count per family:

In our code, we have created a data pipeline which combines two different sub pipelines which handle the categorical and numerical features respectively. Note that we have used inline method of combining the numerical and categorical pipelines. This is shown in the figure below.

```
data_pipeline = make_column_transformer( #Level 2
    (make_pipeline(SimpleImputer(strategy = 'median'), StandardScaler()),      numerical_features), #Level 3
    (make_pipeline(SimpleImputer(strategy='most_frequent'),
                   OneHotEncoder(handle_unknown='ignore')), categorical_features)
  )
```

Figure 13: Numerical and Categorical pipelines

9.2 Visualization of the modeling pipeline:

The base pipeline from the above image is is then used in a different pipeline where machine learning algorithms are applied on them.

The missing values can be handled by using various strategies like mean, median, constant or most frequent in simple imputer as in the above image. The main function of simple imputer is to replace missing values using a descriptive statistic (e.g. mean, median, or most frequent) along each column, or using a constant value.

```
clf_pipe = make_pipeline(
    data_pipeline,
    LogisticRegression())
```

Figure 14: Code for pipeline

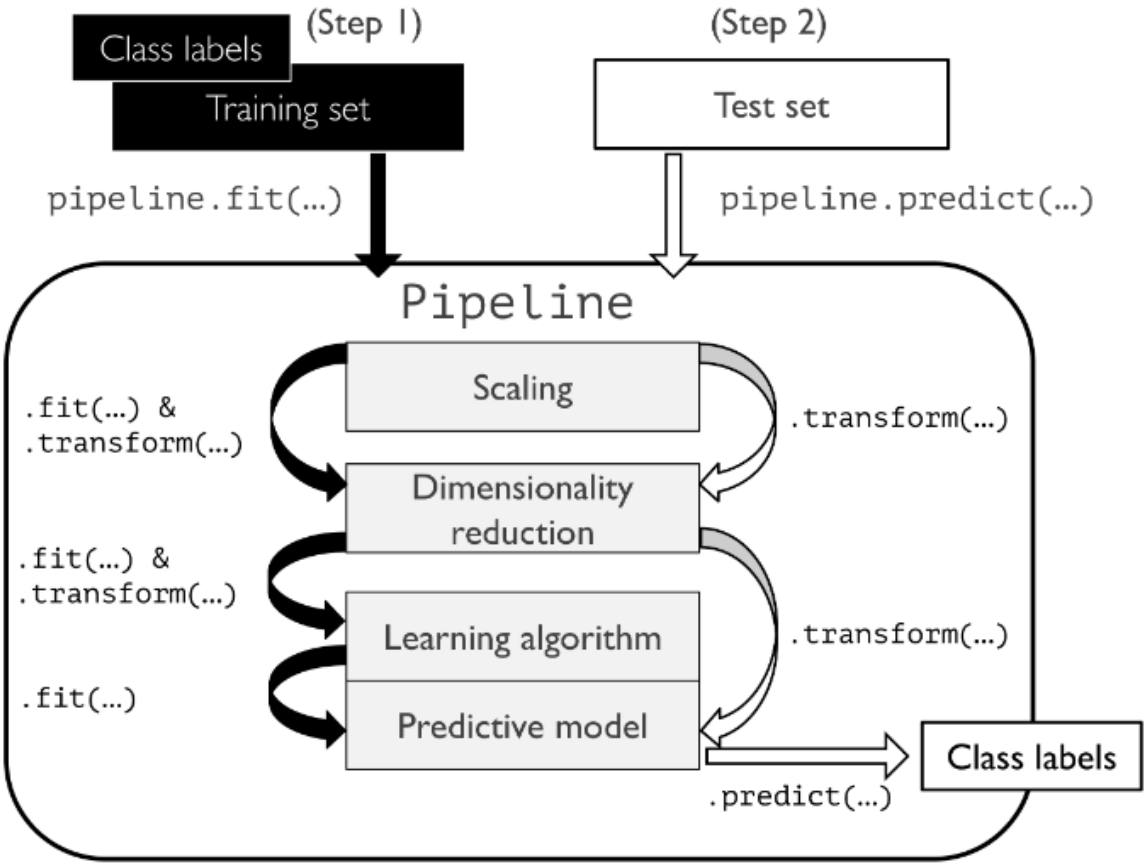


Figure 15: Overall machine learning pipeline in sklearn

9.3 Hyperparameters and settings considered

For the algorithms chosen the parameters chosen by us are as below -

```
grid_params_ab = [{'clf__n_estimators':[100,200],
          'clf__learning_rate':[0.01, 0.1, 1]}]

jobs = -1

AB = GridSearchCV(estimator=pipe_ab,
            param_grid=grid_params_ab,
            scoring='accuracy',
            cv=5,
            n_jobs=jobs)
```

```
### Decision tree

param_range_dt = [3, 6]
grid_params_dt = [{'clf__criterion': ['gini', 'entropy'],
        'clf__max_depth': param_range_dt,
        'clf__min_samples_split': param_range[1:]}]

DT = GridSearchCV(estimator=pipe_dt,
            param_grid=grid_params_dt,
            scoring='accuracy',
            cv=5,
            n_jobs=jobs)
```

```
### MLP


grid_params_mlp = [{'clf__activation': ['relu', 'tanh', 'logistic', 'identity','softmax'],
          'clf__hidden_layer_sizes': [(100,), (50,100,), (50,75,100,)],
          'clf__solver': ['adam', 'sgd', 'lbfgs'],
          'clf__learning_rate' : ['constant', 'adaptive', 'invscaling']}]

jobs = -1

MLP= GridSearchCV(estimator=pipe_mlp,
           param_grid=grid_params_mlp,
           cv=3,
           n_jobs=jobs, verbose=5)
```

```
grid_params_rf = [{'clf__criterion': ['gini', 'entropy'],
        'clf__max_depth': param_range,
        'clf__min_samples_split': param_range[1:]}]

jobs = -1

RF = GridSearchCV(estimator=pipe_rf,
            param_grid=grid_params_rf,
            scoring='accuracy',
            cv=5,
            n_jobs=jobs)
```

```
grid_params_xgb = {'clf__n_estimators':[100,200],
          'clf__learning_rate':[0.01, 0.1, 1]}

jobs = -1

XGB = GridSearchCV(estimator=pipe_xgb,
             param_grid=grid_params_xgb,
             scoring='roc_auc',
             cv=3,
             n_jobs=jobs)
```

9.4 Number of experiments conducted and results:

After calculation the accuracy scores on training, test and validation datasets, all these calculated scores are displayed using logs as in the below snippet

```
try: experimentLog
except : experimentLog = pd.DataFrame(columns=["Pipeline", "Dataset", "TrainAcc", "ValidAcc", "TestAcc", "AUC Train", "AUC Valid", "AUC Test",
                                               "RMSE Train","LogLoss Train","MAE Train","RMSE Test",
                                               "LogLoss Test","MAE Test","RMSE Valid","LogLoss Valid",
                                               "MAE Valid","Train Time(s)", "Test Time(s)"])
experimentLog.loc[len(experimentLog)] =[f"Baseline 1 DecisionTreeClassifier with {number_of_inputs} inputs", "HCDR",
                                        f"{trainAcc*100:8.2f}%", f"{validAcc*100:8.2f}%", f"{testAcc*100:8.2f}%", auc_train, auc_valid, auc_test,
                                        rmse_train,log_loss_train,mae_train,rmse_test,log_loss_test,mae_test,
                                        rmse_valid, log_loss_valid, mae_valid,train_time, test_time]

display(experimentLog)
```

Figure 16: Output log table

9.5 Experiment table

We conducted multiple experiments as in the below snippets -

	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	AUC Train	AUC Valid	AUC Test	RMSE Train	LogLoss Train	MAE Train	RMSE Test	LogLoss Test	MAE Test	RMSE Valid	LogLoss Valid	MAE Valid	Train Time(s)
0	Baseline 1 Logistic Regression with 166 inputs	HCDR	91.92%	91.95%	91.94%	0.50488	0.50513	0.50550	0.28427	2.79123	0.08081	0.28395	2.78470	0.08063	0.28378	2.78150	0.08053	34.4880
1	Baseline 1 DecisionTreeClassifier with 166 inputs	HCDR	92.02%	91.87%	91.85%	0.51123	0.50500	0.50476	0.28249	2.75613	0.07980	0.28555	2.81615	0.08154	0.28510	2.80733	0.08128	20.7824
2	Baseline 1 Perceptron with 166 inputs	HCDR	90.62%	90.63%	90.66%	0.50777	0.50757	0.50791	0.30620	3.23841	0.09376	0.30553	3.22424	0.09335	0.30606	3.23527	0.09367	7.4847
3	Baseline 1 SGDClassifier with 166 inputs	HCDR	91.92%	91.95%	91.94%	0.50000	0.50009	0.50012	0.28424	2.79053	0.08079	0.28387	2.78320	0.08058	0.28367	2.77925	0.08047	13.7471
4	Baseline 1 RandomForestClassifier with 166 inputs	HCDR	91.92%	91.95%	91.94%	0.50000	0.50000	0.50000	0.28424	2.79053	0.08079	0.28387	2.78320	0.08058	0.28367	2.77925	0.08047	21.9577

Figure 17: Experiments conducted

We get output as in the below snippet.

	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	AUC Train	AUC Valid	AUC Test	RMSE Train	LogLoss Train	MAE Train	RMSE Test	LogLoss Test	MAE Test	RMSE Valid	LogLoss Valid	MAE Valid	Train Time(s)	Test Time(s)
0	Baseline 1 Logistic Regression with 193 inputs	HCDR	91.92%	91.94%	91.93%	0.50511	0.50507	0.50545	0.28425	2.79081	0.0808	0.2841	2.78769	0.08071	0.28399	2.78543	0.08065	4.3951	0.144

Figure 18: Output log

9.6 Loss function and other equation metrics used in Latex

$$MAE = \sum_{i=1}^{D} |x_i - y_i| \tag{1}$$

$$MSE = \sum_{i=1}^{D} (x_i - y_i)^2 \tag{2}$$

$$RMSE = (\sqrt{\frac{1}{n}\sum_{n=1}^{N} u^2}) \tag{3}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

$$LogLoss = np.round(logLoss(yValid, ypredprobaValid), 5) \tag{5}$$

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

While designing the pipelines, the losses which were accounted for by us were AUC, RMSE, MSE, MAE and log loss. This is as below -

```
### For AUC
from sklearn import metrics
y_pred_proba_train = clf_pipe.predict(X_train)
auc_train = np.round(metrics.roc_auc_score(y_train, y_pred_proba_train),5)

y_pred_proba_valid = clf_pipe.predict(X_valid)
auc_valid = np.round(metrics.roc_auc_score(y_valid, y_pred_proba_valid),5)

y_pred_proba_test = clf_pipe.predict(X_test)
auc_test = np.round(metrics.roc_auc_score(y_test, y_pred_proba_test),5)
```

Figure 19: AUC

```
## Train loss calculations
mse_train = mean_squared_error(y_train, y_pred_proba_train)
rmse_train = np.sqrt(np.round(mse_train,5))
rmse_train = np.round(rmse_train,5)
print("Train RMSE",rmse_train)

log_loss_train = np.round(log_loss(y_train, y_pred_proba_train),5)
print(" Train LL",log_loss_train)

mae_train = np.round(mean_absolute_error(y_train, y_pred_proba_train),5)
print("Train MAE",mae_train)


## Test Loss Calculation
mse_test = mean_squared_error(y_test, y_pred_proba_test)
rmse_test = np.sqrt(np.round(mse_test,5))
rmse_test = np.round(rmse_test,5)
print("Test RMSE",rmse_test)

log_loss_test = np.round(log_loss(y_test, y_pred_proba_test),5)
print("Test LL",log_loss_test)

mae_test = np.round(mean_absolute_error(y_test, y_pred_proba_test),5)
print("Test MAE",mae_test)

## Validation set Loss Calculation
mse_valid = mean_squared_error(y_valid, y_pred_proba_valid)
rmse_valid = np.sqrt(np.round(mse_valid,5))
rmse_valid = np.round(rmse_valid,5)
print("Valid_RMSE",rmse_valid)

log_loss_valid = np.round(log_loss(y_valid, y_pred_proba_valid),5)
print("LL_valid",log_loss_valid)

mae_valid = np.round(mean_absolute_error(y_valid, y_pred_proba_valid),5)
print("MAE_valid",mae_valid)
```

Figure 20: Output log

10 Hyperparameter tuning

We have used Grid search from sklearn in combination with pipeline in order to perform hyperparameter tuning.In order to achieve this, we have have created multiple pipelines where each pipeline is dedicated to having one of the machine learning algorithms executed as below -

```
###Decision tree classifier
pipe_dt =  Pipeline([('preprocess',data_pipeline),
 ('clf', DecisionTreeClassifier(random_state=42))])

###Random forest classifier
pipe_rf =  Pipeline([('preprocess',data_pipeline),
 ('clf', RandomForestClassifier())])

###Ada boost classifier
pipe_ab =  Pipeline([('preprocess',data_pipeline),
 ('clf',  AdaBoostClassifier())])

### XGB classifier
pipe_xgb =  Pipeline([('preprocess',data_pipeline),
 ('clf',  XGBClassifier())])

### Multi layer perceptron classifier
pipe_mlp = Pipeline([('preprocess',data_pipeline),
 ('clf', MLPClassifier(random_state=42))])
```

Post this, we have created separate grid search chunks for each of these algorithms where parameters are separately defined for the respective algorithms. The GridSearch function from sklearn takes the pipeline, parameters and the number cross validation sets as inputs. Using gridsearch, we can fit the training set data and predict the target variable of the test set. An example chunk for grid search is as below -

```
### Random Forest
grid_params_rf = [{'clf__criterion': ['gini', 'entropy'],
        'clf__max_depth': param_range,
        'clf__min_samples_split': param_range[1:]}]

jobs = -1

RF = GridSearchCV(estimator=pipe_rf,
            param_grid=grid_params_rf,
            scoring='accuracy',
            cv=5,
            n_jobs=jobs)
```

Grid search selects the best parameters by using .bestparams and outputs the values of the parameters which give the best accuracy score. The code chunk and the result are as given below -

```
print('Performing model optimizations...')
best_acc = 0.0
best_clf = 0
best_gs = ''
for idx, gs in enumerate(grids):
    print('\nEstimator: %s' % grid_dict[idx])
    start = time()
    gs.fit(X_train, y_train)
    train_time = np.round(time() - start, 4)
    print('Best params are : %s' % gs.best_params_)
    print('train time : %s' % train_time)
    # Best training data accuracy
    print('Best training accuracy: %.7f' % gs.best_score_)
    # Predict on test data with best params
    start = time()
    y_pred = gs.predict(X_test)
    test_time = np.round(time() - start, 4)
    # Test data accuracy of model with best params
    print('Test set accuracy score for best params: %.7f ' % accuracy_score(y_test, y_pred))
    print('test time : %s' % test_time)
    cv30Splits = ShuffleSplit(n_splits=30 , test_size=0.3,random_state=0)
    logit_scores_valid = cross_val_score(pipe_rf,X_valid,y_valid,cv = cv30Splits)
    best_validation_scores = cross_val_score(gs.best_estimator_, X_valid,y_valid, cv = cv30Splits)
    (t_stat, p_value) = stats.ttest_rel(logit_scores_valid, best_validation_scores)
    results.loc[0] = ["Random Forest",p_value,gs.best_score_ ,accuracy_score(y_test, y_pred), train_time, test_time, gs.best_params_]
    print(results)
    # Track best (highest test accuracy) model
    if accuracy_score(y_test, y_pred) > best_acc:
        best_acc = accuracy_score(y_test, y_pred)
        best_gs = gs
        best_clf = idx

print('\nClassifier with best test set accuracy: %s' % grid_dict[best_clf])
```

10.1 Optimum parameters

Grid search enables us to select best parameters from the set of parameters which are given as input by us. These include learning rate, criteria used, max depth, number of hidden layers in case of perceptron, etc. In case of the example of random forest algorithm which is chosen by us, these optimum parameters are as displayed below.

```
Performing model optimizations...

Estimator: Random Forest
Best params are : {'clf__criterion': 'gini', 'clf__max_depth': 1, 'clf__min_samples_split': 3}
train time : 147.9937
Best training accuracy: 0.9192059
Test set accuracy score for best params: 0.9194181
test time : 0.5536
           ExpID  p-value  Train accuracy  Test Accuracy  Train Time(s)  \
0  Random Forest   0.0579        0.919206       0.919418       147.9937

   Test Time(s)                             Experiment description
0        0.5536  {'clf__criterion': 'gini', 'clf__max_depth': 1...
```

The above chunks of code are repeated for all the machine learning algorithms which have been selected by us.

10.2 Methods chosen - GridSearch

For any business problem, it is always desirable to use multiple algorithms on a given data and finally choose the algorithm which gives the best fit.Also, the performance of a given algorithm can be increased by selecting optimum hyperparameters.Hyperparameters for a model can be chosen using several techniques such as Random Search, Grid Search, Manual Search, Bayesian Optimizations, etc.Grid Search uses a different combination of all the specified hyperparameters and their values and calculates the performance for each combination and selects the best value for the hyperparameters. This makes the processing time-consuming and expensive based on the number of hyperparameters involved.In GridSearchCV, along with Grid Search, cross-validation is also performed. Cross-Validation is used while training the model. As we know that before training the model with data, we divide the data into two parts – train data and test data. In cross-validation, the process divides the train data further into two parts – the train data and the validation data. Due to all these advantages of gridsearch, we have selected this method for phase 3.

11 Neural Networks

11.1 Implementation

Normally, we consider a neural network structure given below. Here, in the given figure we see a hidden layer count of two for better understanding and simplicity, but as the complexity of the model increases, the number of hidden layers also increase respectively. Every neuron has an activation function which feeds inputs to the next layer neurons. The main benefit of neural networks comes from its optimal backpropagation strategy where it is termed as model learning in which the weights are adjusted at every iteration to yield a better output. The weight updations can be seen by the formula: f(x,x,x) = w*x + w*x + w*x + b. There are various activation functions for every neuron, most popular are which Tanh, ReLu, Leaky ReLu, and Softmax. In our model, we have experimented with varied amounts of hidden layers with activation functions of Softmax, ReLu and combinations of both.

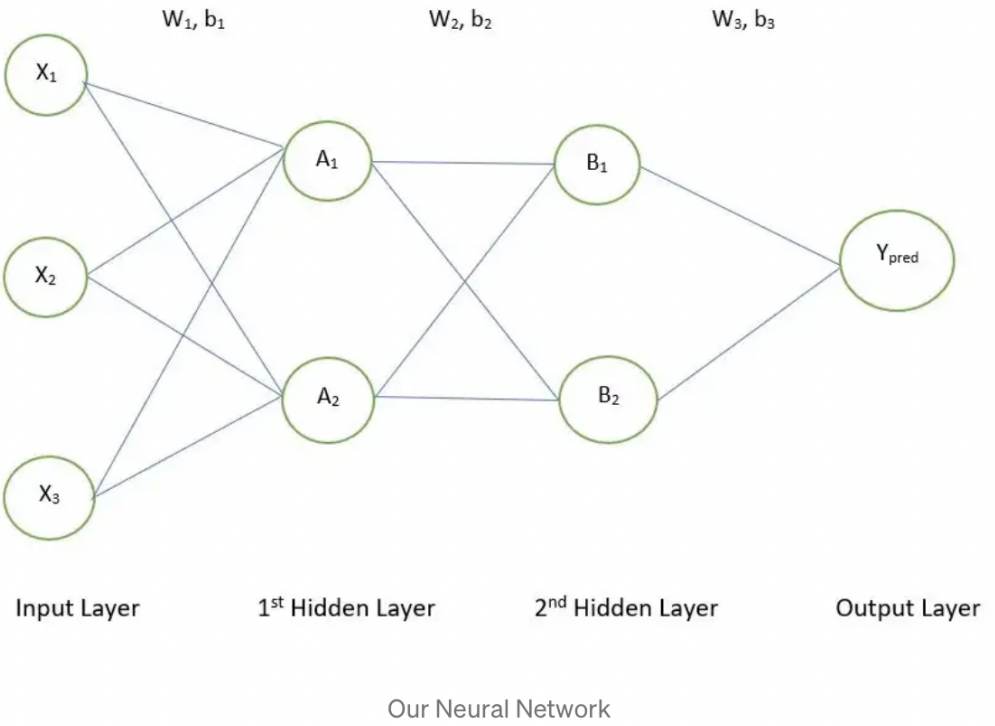


Figure 21: Neural Network Architecture

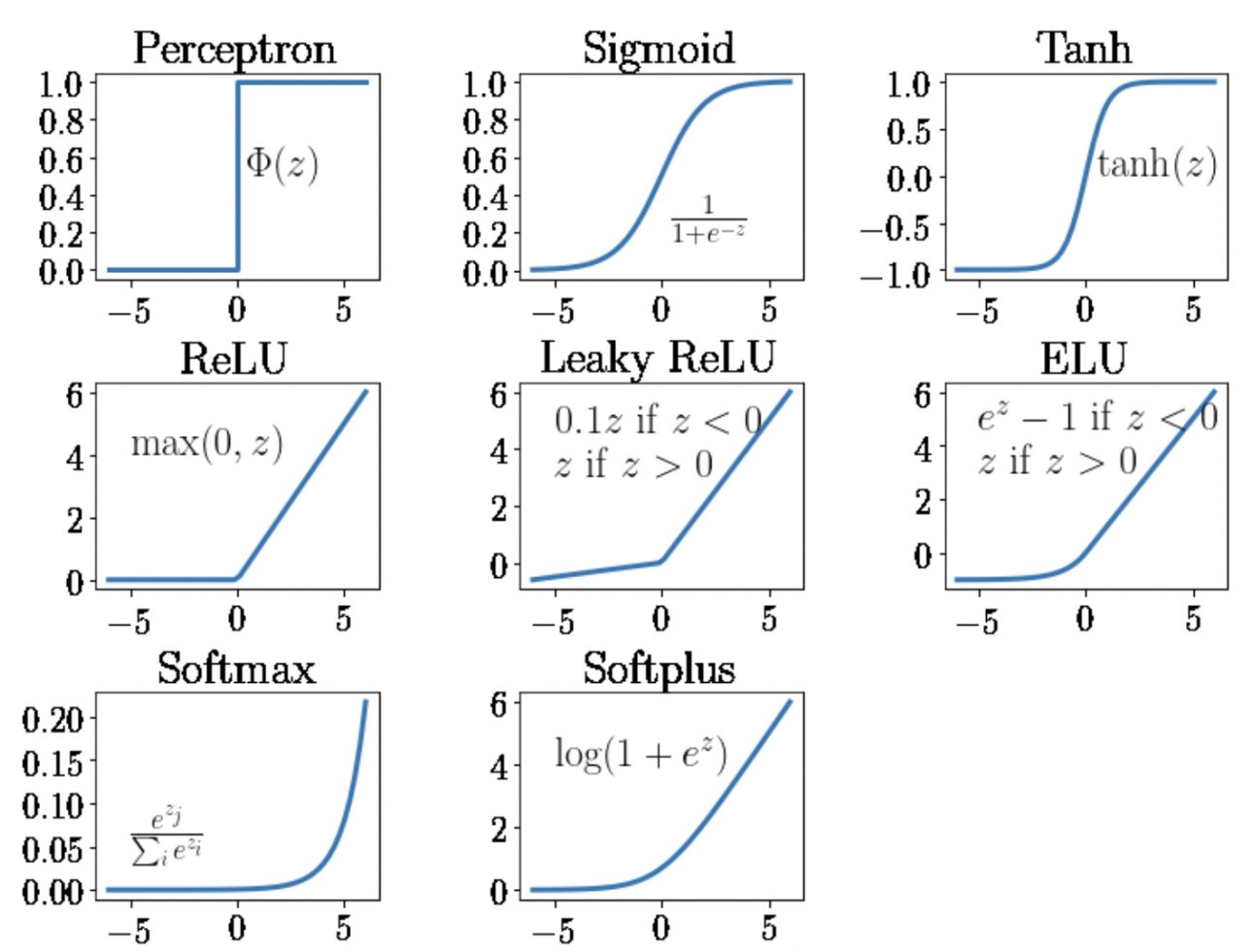


Figure 22: Activation Functions and their graph interpretations

11.2 Network Architecture Experimentation

11.2.1 Architecture 1 (Only ReLU)

For this architecture, we have used a series of 4 hidden layers with 165 input features and 2 output architecture as our model is binary. The activation function used is Rectified Linear Unit for all hidden layers. For the first hidden layer we have in-features = 165 and out-features = 32, with the 2nd layer consisting of in-features = 32 and out-features being 16 and so on til the last hidden layer yields a binary classification with output of 2 features.

11.2.2 Architecture 2 (Only Softmax)

For this architecture, we have used a series of 4 hidden layers with 165 input features and 2 output architecture as our model is binary. The activation function used is Softmax for all hidden layers. For the first hidden layer we have in-features = 165 and out-features = 32, with the 2nd layer consisting of in-features = 32 and out-features being 16 and so on til the last hidden layer yields a binary classification with output of 2 features.

11.2.3 Architecture 3 (Amalgamation of ReLu and Softmax)

This architecture, we have an Amalgamation ReLu and Softmax activation functions. Here, we have use experimented with the input layer utilizing a softmax function and the rest hidden layers comprising of ReLu

activation functions. These layers consist of 4 hidden layers with 165 input features and 2 output architecture as our model is binary. The activation function used is Rectified Linear Unit for all hidden layers. For the first hidden layer we have in-features = 165 and out-features = 32, with the 2nd layer consisting of in-features = 32 and out-features being 16 and so on til the last hidden layer yields a binary classification with output of 2 features.

We also made another Amalgamation with the same Architecture which only uses Softmax for the output layer.

This Architecture below belongs to Architecture group 1 and contains activation function ReLu with loss function of Cross Entropy. This has been run for both SGD and Adam optimizers.
The String Configuration is: 165-64-ReLu 64-32-ReLu 32-16-ReLu 16-8-ReLu 8-2 (Bias = True)

```
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=165, out_features=64, bias=True)
  (1): ReLU()
  (2): Linear(in_features=64, out_features=32, bias=True)
  (3): ReLU()
  (4): Linear(in_features=32, out_features=16, bias=True)
  (5): ReLU()
  (6): Linear(in_features=16, out_features=8, bias=True)
  (7): ReLU()
  (8): Linear(in_features=8, out_features=2, bias=True)
)
--------------------------------------------------
Loss Function: CXE

Epoch 1: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.364       Valid Loss:0.288
Epoch 2: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.283       Valid Loss:0.28
Epoch 3: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 4: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.28        Valid Loss:0.28
Epoch 5: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.28        Valid Loss:0.279
Epoch 6: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.28        Valid Loss:0.279
Epoch 7: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.28        Valid Loss:0.279
Epoch 8: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.279       Valid Loss:0.278
Epoch 9: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.279       Valid Loss:0.278
Epoch 10: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.278       Valid Loss:0.277
Epoch 11: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.278       Valid Loss:0.276
Epoch 12: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.277       Valid Loss:0.276
Epoch 13: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.276       Valid Loss:0.274
Epoch 14: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.275       Valid Loss:0.273
Epoch 15: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.273       Valid Loss:0.272
Epoch 16: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.272       Valid Loss:0.27
Epoch 17: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.271       Valid Loss:0.269
Epoch 18: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.269       Valid Loss:0.268
Epoch 19: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.268       Valid Loss:0.267
Epoch 20: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.267       Valid Loss:0.266
Epoch 21: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.266       Valid Loss:0.265
Epoch 22: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.265       Valid Loss:0.264
Epoch 23: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.265       Valid Loss:0.263
Epoch 24: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.264       Valid Loss:0.263
Epoch 25: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.264       Valid Loss:0.262
--------------------------------------------------
```

Out[17]:

	Architecture string	Optimizer	Epochs	Train accuracy	Valid accuracy	Test accuracy	Test Loss
0	165-64-32-16-8-2	<class 'torch.optim.adam.Adam'>	25	92.4%	91.4%	91.4%	0.274%
1	165-64-32-16-8-2	<class 'torch.optim.sgd.SGD'>	25	91.9%	92.0%	91.9%	0.263%

Figure 23: ReLu and CXE (Architecture 1)

This Architecture below belongs to Architecture group 1 and contains activation function ReLu with loss function of Cross Entropy and Mean Squared Error. This has been run for both SGD optimizer.
The String Configuration is: 165-64-ReLu 64-32-ReLu 32-16-ReLu 16-8-ReLu 8-2 (Bias = True)

```
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=165, out_features=64, bias=True)
  (1): ReLU()
  (2): Linear(in_features=64, out_features=32, bias=True)
  (3): ReLU()
  (4): Linear(in_features=32, out_features=16, bias=True)
  (5): ReLU()
  (6): Linear(in_features=16, out_features=8, bias=True)
  (7): ReLU()
  (8): Linear(in_features=8, out_features=2, bias=True)
)
--------------------------------------------------
Loss Fumction: CXE + MSE
Epoch 1: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.663       Valid Loss:0.65
Epoch 2: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.65
Epoch 3: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 4: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 5: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 6: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.649       Valid Loss:0.649
Epoch 7: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.649       Valid Loss:0.649
Epoch 8: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.649       Valid Loss:0.649
Epoch 9: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.649       Valid Loss:0.649
Epoch 10: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.649       Valid Loss:0.649
Epoch 11: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.649       Valid Loss:0.649
Epoch 12: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.649       Valid Loss:0.648
Epoch 13: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.649       Valid Loss:0.648
Epoch 14: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.648       Valid Loss:0.648
Epoch 15: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.648       Valid Loss:0.648
Epoch 16: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.648       Valid Loss:0.648
Epoch 17: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.648       Valid Loss:0.647
Epoch 18: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.648       Valid Loss:0.647
Epoch 19: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.647       Valid Loss:0.647
Epoch 20: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.647       Valid Loss:0.647
Epoch 21: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.647       Valid Loss:0.646
Epoch 22: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.647       Valid Loss:0.646
Epoch 23: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.646       Valid Loss:0.646
Epoch 24: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.646       Valid Loss:0.646
Epoch 25: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.646       Valid Loss:0.646
--------------------------------------------------
```

Out[16]:

	Architecture string	Optimizer	Epochs	Train accuracy	Valid accuracy	Test accuracy	Test Loss
0	165-64-32-16-8-2	<class 'torch.optim.sgd.SGD'>	25	91.9%	92.0%	91.9%	0.646%

Figure 24: ReLu and CXE + MSE (Architecture 1)

This Architecture below belongs to Architecture group 2 and contains activation function Softmax with loss function of Cross Entropy. This has been run for both SGD and Adam optimizers.
The String Configuration is: 165-64-Softmax 64-32-Softmax 32-16-Softmax 16-8-Softmax 8-2 (Bias = True)

```
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=165, out_features=64, bias=True)
  (1): Softmax(dim=None)
  (2): Linear(in_features=64, out_features=32, bias=True)
  (3): Softmax(dim=None)
  (4): Linear(in_features=32, out_features=16, bias=True)
  (5): Softmax(dim=None)
  (6): Linear(in_features=16, out_features=8, bias=True)
  (7): Softmax(dim=None)
  (8): Linear(in_features=8, out_features=2, bias=True)
)
--------------------------------------------------
Loss Function: CXE
Epoch 1: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.307
Epoch 2: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.293       Valid Loss:0.285
Epoch 3: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.283       Valid Loss:0.281
Epoch 4: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 5: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 6: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 7: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 8: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 9: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 10: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 11: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 12: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 13: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 14: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 15: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 16: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 17: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 18: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 19: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 20: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 21: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 22: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 23: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 24: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 25: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
--------------------------------------------------
```

Out[17]:

	Architecture string	Optimizer	Epochs	Train accuracy	Valid accuracy	Test accuracy	Test Loss
0	165-64-32-16-8-2	<class 'torch.optim.adam.Adam'>	25	91.9%	91.8%	91.8%	0.267%
1	165-64-32-16-8-2	<class 'torch.optim.sgd.SGD'>	25	91.9%	92.0%	91.9%	0.28%

Figure 25: Softmax and CXE (Architecture 2)

This Architecture below belongs to Architecture 2 and contains activation function Softmax with loss function of Cross Entropy and Mean Squared Error. This has been run for both SGD and Adam optimizers. The String Configuration is: 165-64-Softmax 64-32-Softmax 32-16-Softmax 16-8-Softmax 8-2

```
-----------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=165, out_features=64, bias=True)
  (1): Softmax(dim=None)
  (2): Linear(in_features=64, out_features=32, bias=True)
  (3): Softmax(dim=None)
  (4): Linear(in_features=32, out_features=16, bias=True)
  (5): Softmax(dim=None)
  (6): Linear(in_features=16, out_features=8, bias=True)
  (7): Softmax(dim=None)
  (8): Linear(in_features=8, out_features=2, bias=True)
)
-----------------------------------------------------
Loss Fumction: CXE + MSE
Epoch 1: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.668       Valid Loss:0.649
Epoch 2: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 3: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 4: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 5: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 6: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 7: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 8: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 9: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 10: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 11: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 12: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 13: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 14: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 15: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 16: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 17: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 18: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 19: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 20: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 21: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 22: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 23: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 24: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 25: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
-----------------------------------------------------
```

Out[17]:

	Architecture string	Optimizer	Epochs	Train accuracy	Valid accuracy	Test accuracy	Test Loss
0	165-64-32-16-8-2	<class 'torch.optim.adam.Adam'>	25	92.2%	91.4%	91.4%	0.648%
1	165-64-32-16-8-2	<class 'torch.optim.sgd.SGD'>	25	91.9%	92.0%	91.9%	0.649%

Figure 26: Softmax and MSE + CXE (Architecture 2)

This Architecture below belongs to Architecture group 3 contains activation function ReLu and Softmax with loss function of Cross Entropy and Mean Squared Error. This has been run for both SGD and Adam optimizers. The initial layer has an activation function of Softmax and the hidden layers + the output layers have ReLu as their activation.

The String Configuration is: 165-64-Softmax 64-32-ReLu 32-16-ReLu 16-8-ReLu 8-2

```
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=165, out_features=64, bias=True)
  (1): Softmax(dim=None)
  (2): Linear(in_features=64, out_features=32, bias=True)
  (3): ReLU()
  (4): Linear(in_features=32, out_features=16, bias=True)
  (5): ReLU()
  (6): Linear(in_features=16, out_features=8, bias=True)
  (7): ReLU()
  (8): Linear(in_features=8, out_features=2, bias=True)
)
--------------------------------------------------
Loss Fumction: CXE + MSE
Epoch 1: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.647       Valid Loss:0.643
Epoch 2: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.644       Valid Loss:0.643
Epoch 3: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.643       Valid Loss:0.643
Epoch 4: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.643       Valid Loss:0.643
Epoch 5: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.642       Valid Loss:0.643
Epoch 6: Train Accuracy: 0.919   valid Accuracy: 0.919   Train Loss: 0.642       Valid Loss:0.643
Epoch 7: Train Accuracy: 0.919   valid Accuracy: 0.919   Train Loss: 0.642       Valid Loss:0.643
Epoch 8: Train Accuracy: 0.919   valid Accuracy: 0.919   Train Loss: 0.641       Valid Loss:0.644
Epoch 9: Train Accuracy: 0.92    valid Accuracy: 0.919   Train Loss: 0.641       Valid Loss:0.644
Epoch 10: Train Accuracy: 0.92   valid Accuracy: 0.919   Train Loss: 0.64        Valid Loss:0.644
Epoch 11: Train Accuracy: 0.92   valid Accuracy: 0.919   Train Loss: 0.64        Valid Loss:0.644
Epoch 12: Train Accuracy: 0.921  valid Accuracy: 0.919   Train Loss: 0.639       Valid Loss:0.644
Epoch 13: Train Accuracy: 0.921  valid Accuracy: 0.917   Train Loss: 0.639       Valid Loss:0.646
Epoch 14: Train Accuracy: 0.921  valid Accuracy: 0.917   Train Loss: 0.638       Valid Loss:0.646
Epoch 15: Train Accuracy: 0.922  valid Accuracy: 0.916   Train Loss: 0.638       Valid Loss:0.646
Epoch 16: Train Accuracy: 0.922  valid Accuracy: 0.916   Train Loss: 0.637       Valid Loss:0.647
Epoch 17: Train Accuracy: 0.922  valid Accuracy: 0.916   Train Loss: 0.637       Valid Loss:0.647
Epoch 18: Train Accuracy: 0.923  valid Accuracy: 0.917   Train Loss: 0.637       Valid Loss:0.646
Epoch 19: Train Accuracy: 0.923  valid Accuracy: 0.914   Train Loss: 0.636       Valid Loss:0.648
Epoch 20: Train Accuracy: 0.923  valid Accuracy: 0.914   Train Loss: 0.636       Valid Loss:0.648
Epoch 21: Train Accuracy: 0.924  valid Accuracy: 0.914   Train Loss: 0.636       Valid Loss:0.648
Epoch 22: Train Accuracy: 0.924  valid Accuracy: 0.913   Train Loss: 0.635       Valid Loss:0.65
Epoch 23: Train Accuracy: 0.924  valid Accuracy: 0.915   Train Loss: 0.635       Valid Loss:0.648
Epoch 24: Train Accuracy: 0.924  valid Accuracy: 0.914   Train Loss: 0.635       Valid Loss:0.649
Epoch 25: Train Accuracy: 0.925  valid Accuracy: 0.913   Train Loss: 0.635       Valid Loss:0.65
--------------------------------------------------
```

Out[17]:

	Architecture string	Optimizer	Epochs	Train accuracy	Valid accuracy	Test accuracy	Test Loss
0	165-64-32-16-8-2	<class 'torch.optim.sgd.SGD'>	25	91.9%	92.0%	91.9%	0.649%
1	165-64-32-16-8-2	<class 'torch.optim.adam.Adam'>	25	92.5%	91.3%	91.3%	0.65%

Figure 27: ReLu + Softmax and MSE + CXE (Architecture 3)

This Architecture below belongs to Architecture group 3 contains activation function ReLu and Softmax with loss function of Cross Entropy. This has been run for both Adam optimizer. The initial layer has an activation function of ReLu and the hidden layers + the output layers have Softmax as their activation.
The String Configuration is: 165-128-ReLU 128-128-ReLu 128-128-ReLu 128-128-ReLu 128-128-ReLu 128-2-Softmax

```
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=165, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=128, bias=True)
  (3): ReLU()
  (4): Linear(in_features=128, out_features=128, bias=True)
  (5): ReLU()
  (6): Linear(in_features=128, out_features=128, bias=True)
  (7): ReLU()
  (8): Linear(in_features=128, out_features=2, bias=True)
  (9): Softmax(dim=None)
)
--------------------------------------------------
Loss Function: CXE

Epoch 1: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.395       Valid Loss:0.394
Epoch 2: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 3: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 4: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 5: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 6: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 7: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 8: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 9: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 10: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 11: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 12: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 13: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 14: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 15: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 16: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 17: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 18: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 19: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 20: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
--------------------------------------------------
0.5
```

Out[15]:

	Architecture string	Optimizer	Epochs	Train accuracy	Valid accuracy	Test accuracy	Test Loss
0	165-128-128-128-128-2	<class 'torch.optim.adam.Adam'>	20	91.9%	92.0%	91.9%	0.394

Figure 28: ReLu + Softmax and CXE (Architecture 3)

11.3 Networks Architecture Configurations and Interpretation

From the above architecture configurations, we get the results as following. The result log suggests that the best activation function is for combination of ReLu and Softmax for the output layer with configuration performance as 91.9 percent with slight better performance with Adam optimizer. This particular configuration of 165-128-ReLu 128-128-ReLu 128-128-ReLu 128-128-ReLu 128-2-Softmax (Bias = True) also yields a better loss as compared to other configurations.

11.4 Configuration Difficulties and Strategies used to Tackle them

In the following Architectures, we used a total of 3 parameters to mix and match with the configurations. The main conundrum we faced was the amalgamation of the loss functions of Cross Entropy and Mean Squared Error. After research and development, the root of this quandary was the difference in their tensor dimensions which acted as a deterrent to their amalgamation. We were forced to modify their tensor dimensionalities to achieve a CXE + MSE loss function to meet the requirement.

12 Classification

In Classification, we used the Sequential model to train our dataset. From the following Architectures, we experimented many combinations with optimizers changing from Adam to SGD. We also implemented loss function of CXE, MSE and MSE + CSE by altering the tensor dimensions in our classification model. Our best solution was obtained in Architecture 3 using Relu for all hidden layers and Softmax for the output layer. In this combination, we only used Cross Entropy as our loss function with optimizer as Adam in our classification model.

```
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=165, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=128, bias=True)
  (3): ReLU()
  (4): Linear(in_features=128, out_features=128, bias=True)
  (5): ReLU()
  (6): Linear(in_features=128, out_features=128, bias=True)
  (7): ReLU()
  (8): Linear(in_features=128, out_features=2, bias=True)
  (9): Softmax(dim=None)
)
--------------------------------------------------
Loss Function: CXE

Epoch 1: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.395       Valid Loss:0.394
Epoch 2: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 3: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 4: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 5: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 6: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 7: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 8: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 9: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 10: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 11: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 12: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 13: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 14: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 15: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 16: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 17: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 18: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 19: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 20: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
--------------------------------------------------
0.5
```

Out[15]:

	Architecture string	Optimizer	Epochs	Train accuracy	Valid accuracy	Test accuracy	Test Loss
0	165-128-128-128-128-2	<class 'torch.optim.adam.Adam'>	20	91.9%	92.0%	91.9%	0.394

Figure 29: Best MLP model for classification

13 Regression

We performed regression on our training data using pytorch neural networks and mean square error(MSE) as our loss function. The target variable chosen by us for this purpose DAYS_BIRTH which indicates the age of the client in days at the time of borrowing the loan. We decided to chose this feature as it may help us predict the age of the clients who are most likely to borrow loans. In order to implement this code we used mean square error as our loss function.The challenge that we were facing in this chunk of code was that we were getting non numeric values as the training and validation MSEs. In order to handle this, we used the following code torch.nn.utils.clip_grad_norm_(model.parameters(),5).

Finally, we got the MSE for training and test dataset as below -

	Architecture string	Optimizer	Epochs	Train MSE	Validation MSE	Test MSE
0	165-32-16-2-1	<class 'torch.optim.sgd.SGD'>	5	98744.596	170.391	170.598

Figure 30: Result for regression using MSE

While performing the regression, we have created a model which has an input layer, three hidden layers and one output layer. The structure of the model is as below -

```
Model:
BaseModel(
  (fc1): Linear(in_features=165, out_features=32, bias=True)
  (intermediate_layers): ModuleList(
    (0): Linear(in_features=32, out_features=16, bias=True)
    (1): Linear(in_features=16, out_features=2, bias=True)
  )
  (fc_output): Linear(in_features=2, out_features=1, bias=True)
)
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 32]           5,312
            Linear-2                [-1, 1, 16]             528
            Linear-3                 [-1, 1, 2]              34
            Linear-4                 [-1, 1, 1]               3
================================================================
Total params: 5,877
Trainable params: 5,877
Non-trainable params: 0
```

Figure 31: Regression MLP model

The log that we could observe after tuning for a few parameters is as below-

[57]:

	Architecture string	Optimizer	Epochs	Train MSE	Validation MSE	Test MSE
0	165-32-16-2-1	<class 'torch.optim.sgd.SGD'>	5	98744.596	170.391	170.598
1	166-32-16-2-1	<class 'torch.optim.adam.Adam'>	5	304.333	2.368	2.37
2	166-32-10-2-1	<class 'torch.optim.adam.Adam'>	5	304.288	2.368	2.37
3	166-20-10-2-1	<class 'torch.optim.sgd.SGD'>	5	304.731	2.38	2.382
4	166-20-10-2-1	<class 'torch.optim.adam.Adam'>	5	304.646	2.365	2.367

Figure 32: Regression MLP Logs

We implemented this model using OOP by taking reference from homework 11 ipynb document for pytorch.

14 Leakage (HCDR)

Data leakage is term referred when the model is already aware of some part of the test data which leads to very good results for the time being. Since in our model, we have a dedicated application test set and our train-test-test split function our model uses the function 'transform' which makes it evident of no leakage in our model.

14.1 Data Leakage

In the previous phase we were tackling Feature Engineering to select the most significant features utilizing feature selection, handling the missing data, data normalization and addition of essential features. In this process of normalization, we brought more significant features such as DAYS-CREDIT, AMT-CREDIT and EXT-SOURCES from other datasets into picture. In addition to this, we only kept the features where 85 percent of the data was available. A more detailed documentation about feature selection and parameter tuning can be found in Phase-3/Phase 2 report. As understood by us from the previously attended lab section,

data leakage happens mostly when the splitting of the training data into training, test and validation datasets is done after performing standardization, normalization, etc on the entire data. This happens because the validation an test data which should remain untouched while training the model gets transformed in the preprocessing step. In order to avoid this, we have ensured that the splitting of the data has been performed before we do preprocessing of the data by using standard scalar or performing one hot encoding.The code demonstrating this is as below:

14.2 Are any cardinal sins being violated? Describe How have we avoided it?

As part of our pipeline creation, ensured that the splitting of the training data into test, validation and training data was strictly done before standardization on the data which was being used to train the model. Thus, we avoided any leakage from the held out set and also we created the pipelines using transform method to ensure that there is no data leakage and no cardinal sin is violated.

```
# Split the provided training data into training and validationa and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"X train           shape: {X_train.shape}")
print(f"X validation      shape: {X_valid.shape}")

# determine categorical and numerical features
numerical_features = list(X.select_dtypes(include=['int64', 'float64']).columns)
categorical_features = list(X.select_dtypes(include=['object', 'bool']).columns)
print(f"numerical_features are  : {numerical_features}")
print(f"categorical_features are: {categorical_features}")

data_pipeline = make_column_transformer( #Level 2
    (make_pipeline(SimpleImputer(strategy = 'median'), StandardScaler()),      numerical_features), #Level 3
    (make_pipeline(SimpleImputer(strategy='most_frequent'),
                   OneHotEncoder(handle_unknown='ignore')), categorical_features)
  )
```

Figure 33: Pipelining in order to avoid leakage

15 Gap Analysis

- From our earlier results, we had an overal AUC score of 0.5 with Logistic Regression. By using tree data structures like Decision Tree Classifier with proper EDA and feature selection, our AUC score improved to 0.7 which in noted in Kaggle scoreboard.
- The success from Decision Trees made it evident to follow more tree based algorithms such as XGBoost and Random Forests. By using Pytorch on MLP, although we experimented with many Architectures and combinations with our significantly improved execution time, our AUC score plummeted down to 0.5 which suggested that Decision Tree was a better performer in this aspect.
- Also, our test loss had an improvement over the older models which contributed to the better performance.
- Also, in the current phase, we initially had a simple output layer which was in turn modified to Softmax which was also a performance boost to our model.

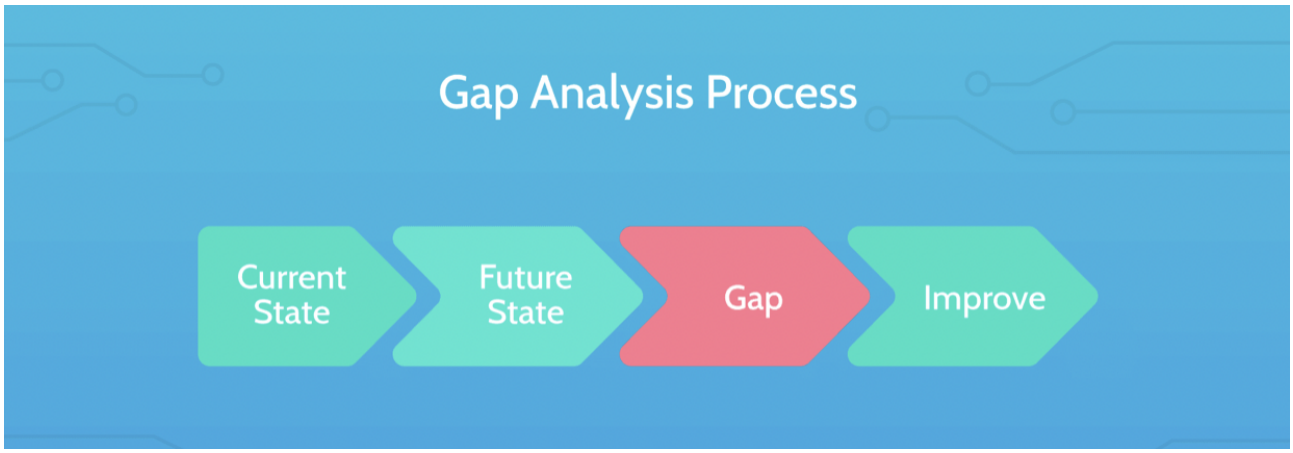


Figure 34: Gap Analysis

16 Results and Discussion of Results

16.1 Pytorch Classification Results based on Architecture

In Classification, we used the Sequential model to train our dataset. From the following Architectures, we experimented many combinations with optimizers changing from Adam to SGD. We also implemented loss function of CXE, MSE and MSE + CSE by altering the tensor dimensions in our classification model. We first checked the first testing of the architecture model on Tensor board to check the curves and analyze some significant features from it. We analyzed Tensor curves which were plotted for Classification and Regression MSE, CXE. Over there, we extracted insights with the approach of dropping some features which had a uniform or lower valued curve as compared to other features. This reduced our Train and validation loss to some extent and made our model more accurate. Further, we got to a conclusion that, our best solution was obtained in Architecture 3 using Relu for all hidden layers and Softmax for the output. In this combination, we only used Cross Entropy as our loss function with optimizer as Adam in our classification model. The particular configuration of 165-128-ReLu 128-128-ReLu 128-128-ReLu 128-128-ReLu 128-2-Softmax (Bias = True yielded best output for the model.

```
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=165, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=128, bias=True)
  (3): ReLU()
  (4): Linear(in_features=128, out_features=128, bias=True)
  (5): ReLU()
  (6): Linear(in_features=128, out_features=128, bias=True)
  (7): ReLU()
  (8): Linear(in_features=128, out_features=2, bias=True)
  (9): Softmax(dim=None)
)
--------------------------------------------------
Loss Function: CXE

Epoch 1: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.395       Valid Loss:0.394
Epoch 2: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 3: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 4: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 5: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 6: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 7: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 8: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 9: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 10: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 11: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 12: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 13: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 14: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 15: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 16: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 17: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 18: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 19: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 20: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
--------------------------------------------------
0.5
```

Out[15]:

	Architecture string	Optimizer	Epochs	Train accuracy	Valid accuracy	Test accuracy	Test Loss
0	165-128-128-128-128-2	<class 'torch.optim.adam.Adam'>	20	91.9%	92.0%	91.9%	0.394

Figure 35: Best MLP model for classification

The log that we could observe for Regression after tuning for a few parameters is as below-

[57]:

	Architecture string	Optimizer	Epochs	Train MSE	Validation MSE	Test MSE
0	165-32-16-2-1	<class 'torch.optim.sgd.SGD'>	5	98744.596	170.391	170.598
1	166-32-16-2-1	<class 'torch.optim.adam.Adam'>	5	304.333	2.368	2.37
2	166-32-10-2-1	<class 'torch.optim.adam.Adam'>	5	304.288	2.368	2.37
3	166-20-10-2-1	<class 'torch.optim.sgd.SGD'>	5	304.731	2.38	2.382
4	166-20-10-2-1	<class 'torch.optim.adam.Adam'>	5	304.646	2.365	2.367

Figure 36: Regression MLP Logs

16.2 Grid Search Results Interpretation

Initially in Grid Search Hyper-parameter Tuning, we utilized Random Forest algorithm with the pipelining where we achieved the best test accuracy as 0.919 with training time as 96.72 secs, p-value as 1.33e-01 and we received the best parameters on 'gini' criterion, max depth as 1 and min samples split as 3. With respect to tree algorithms, we checked Decision Tree algorithm where we are getting a lesser p-value as 5.98e-01 with test accuracy as 0.91 on gini criterion. In order to maximize the kaggle public and private score, we figured out that we have to deep dive into other methods in Tree algorithms due to which we implemented Ada Boost where we achieved the best public and private scores as 0.726 and 0.716 respectively.

	ExpID	p-value	Train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Random Forest	1.339725e-01	0.919206	0.919418	96.7233	0.4367	{'clf__criterion': 'gini', 'clf__max_depth': 1...
1	Decision Tree	5.983691e-44	0.919206	0.919418	99.9313	0.5618	{'clf__criterion': 'gini', 'clf__max_depth': 1...
2	ADA Boost	1.061639e-01	0.919316	0.919592	1812.7945	1.2117	{'clf__learning_rate': 1, 'clf__n_estimators':...
3	MLP	5.562369e-33	0.919247	0.919440	815.5116	0.3444	{'clf__activation': 'relu', 'clf__hidden_layer...

Figure 37: Result table

16.3 PyTorch Results

The activation function is an integral part of a neural network as it provides non-linearity and allows the neural networks to develop complex representations and not just linear relations. So, we have implemented Neural network model with 2 activation architectural functions: 1. Softmax 2. ReLu

Here, we have made use of metrics combinations CXE and combination of MSE and CXEon Softmax and ReLu activation Functions.Further, we analyzed Pytorch curves which were plotted for Classification and Regression MSE, CXE. Over there, we extracted insights with the approach of dropping some features which had a uniform or lower valued curve as compared to other features. This reduced our Train and validation loss to some extent and made our model more accurate.

```
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=165, out_features=64, bias=True)
  (1): ReLU()
  (2): Linear(in_features=64, out_features=32, bias=True)
  (3): ReLU()
  (4): Linear(in_features=32, out_features=16, bias=True)
  (5): ReLU()
  (6): Linear(in_features=16, out_features=8, bias=True)
  (7): ReLU()
  (8): Linear(in_features=8, out_features=2, bias=True)
)
--------------------------------------------------
Loss Fumction: CXE + MSE
Epoch 1: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.663       Valid Loss:0.65
Epoch 2: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.65
Epoch 3: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 4: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 5: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 6: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.649       Valid Loss:0.649
Epoch 7: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.649       Valid Loss:0.649
Epoch 8: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.649       Valid Loss:0.649
Epoch 9: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.649       Valid Loss:0.649
Epoch 10: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.649       Valid Loss:0.649
Epoch 11: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.649       Valid Loss:0.649
Epoch 12: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.649       Valid Loss:0.648
Epoch 13: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.649       Valid Loss:0.648
Epoch 14: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.648       Valid Loss:0.648
Epoch 15: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.648       Valid Loss:0.648
Epoch 16: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.648       Valid Loss:0.648
Epoch 17: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.648       Valid Loss:0.647
Epoch 18: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.648       Valid Loss:0.647
Epoch 19: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.647       Valid Loss:0.647
Epoch 20: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.647       Valid Loss:0.647
Epoch 21: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.647       Valid Loss:0.646
Epoch 22: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.647       Valid Loss:0.646
Epoch 23: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.646       Valid Loss:0.646
Epoch 24: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.646       Valid Loss:0.646
Epoch 25: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.646       Valid Loss:0.646
--------------------------------------------------
```

Out[16]:

	Architecture string	Optimizer	Epochs	Train accuracy	Valid accuracy	Test accuracy	Test Loss
0	165-64-32-16-8-2	<class 'torch.optim.sgd.SGD'>	25	91.9%	92.0%	91.9%	0.646%

Figure 38: Result table

```
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=165, out_features=64, bias=True)
  (1): ReLU()
  (2): Linear(in_features=64, out_features=32, bias=True)
  (3): ReLU()
  (4): Linear(in_features=32, out_features=16, bias=True)
  (5): ReLU()
  (6): Linear(in_features=16, out_features=8, bias=True)
  (7): ReLU()
  (8): Linear(in_features=8, out_features=2, bias=True)
)
--------------------------------------------------
Loss Function: CXE

Epoch 1: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.364       Valid Loss:0.288
Epoch 2: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.283       Valid Loss:0.28
Epoch 3: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.281       Valid Loss:0.28
Epoch 4: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.28        Valid Loss:0.28
Epoch 5: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.28        Valid Loss:0.279
Epoch 6: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.28        Valid Loss:0.279
Epoch 7: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.28        Valid Loss:0.279
Epoch 8: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.279       Valid Loss:0.278
Epoch 9: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.279       Valid Loss:0.278
Epoch 10: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.278       Valid Loss:0.277
Epoch 11: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.278       Valid Loss:0.276
Epoch 12: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.277       Valid Loss:0.276
Epoch 13: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.276       Valid Loss:0.274
Epoch 14: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.275       Valid Loss:0.273
Epoch 15: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.273       Valid Loss:0.272
Epoch 16: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.272       Valid Loss:0.27
Epoch 17: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.271       Valid Loss:0.269
Epoch 18: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.269       Valid Loss:0.268
Epoch 19: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.268       Valid Loss:0.267
Epoch 20: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.267       Valid Loss:0.266
Epoch 21: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.266       Valid Loss:0.265
Epoch 22: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.265       Valid Loss:0.264
Epoch 23: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.265       Valid Loss:0.263
Epoch 24: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.264       Valid Loss:0.263
Epoch 25: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.264       Valid Loss:0.262
--------------------------------------------------
```

Out[17]:

	Architecture string	Optimizer	Epochs	Train accuracy	Valid accuracy	Test accuracy	Test Loss
0	165-64-32-16-8-2	<class 'torch.optim.adam.Adam'>	25	92.4%	91.4%	91.4%	0.274%
1	165-64-32-16-8-2	<class 'torch.optim.sgd.SGD'>	25	91.9%	92.0%	91.9%	0.263%

Figure 39: Result table

```
---------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=165, out_features=64, bias=True)
  (1): Softmax(dim=None)
  (2): Linear(in_features=64, out_features=32, bias=True)
  (3): Softmax(dim=None)
  (4): Linear(in_features=32, out_features=16, bias=True)
  (5): Softmax(dim=None)
  (6): Linear(in_features=16, out_features=8, bias=True)
  (7): Softmax(dim=None)
  (8): Linear(in_features=8, out_features=2, bias=True)
)
---------------------------------------------------
Loss Fumction: CXE + MSE
Epoch 1: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.668       Valid Loss:0.649
Epoch 2: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 3: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 4: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 5: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 6: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 7: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 8: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 9: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 10: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 11: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 12: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 13: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 14: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 15: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 16: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 17: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 18: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 19: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 20: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 21: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 22: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 23: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 24: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
Epoch 25: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.65        Valid Loss:0.649
---------------------------------------------------
```

Out[17]:

	Architecture string	Optimizer	Epochs	Train accuracy	Valid accuracy	Test accuracy	Test Loss
0	165-64-32-16-8-2	<class 'torch.optim.adam.Adam'>	25	92.2%	91.4%	91.4%	0.648%
1	165-64-32-16-8-2	<class 'torch.optim.sgd.SGD'>	25	91.9%	92.0%	91.9%	0.649%

Figure 40: Result table

16.4 Kaggle Scores

We worked more on optimizing the Tree algorithms such that to get a better model producing a best score. We utilizied Decision Tree algorithm with AdaBoost and implemented Multilayer Perceptron using PyTorch with the intention of achieving a best accurate score. We implemented both Classification and Regression using Pytorch with following metrics like MSE, CXE and combination of MSE + CXE with the given achitecture models like Softmax, ReLu. Further, we got Kaggle scores with public score and private score being 0.726 and 0.716 respectively. The best model was Decision Tree algorithm which provided accuracy scale of around 0.92. For getting the kaggle scores, we computed metrics like Mean squared error, found accuracy and precision scale for which we understood how to go around. We had also implemented Feature Importance which assisted us on which insignificant features to be removed. Further, we analyzed Pytorch curves which were plotted for Classification and Regression MSE, CXE. Over there, we extracted insights with the approach of dropping some features which had a uniform or lower valued curve as compared to other features. This reduced our Train and validation loss to some extent and made our model more accurate.

Below is the Kaggle score pictorial depiction of the same.

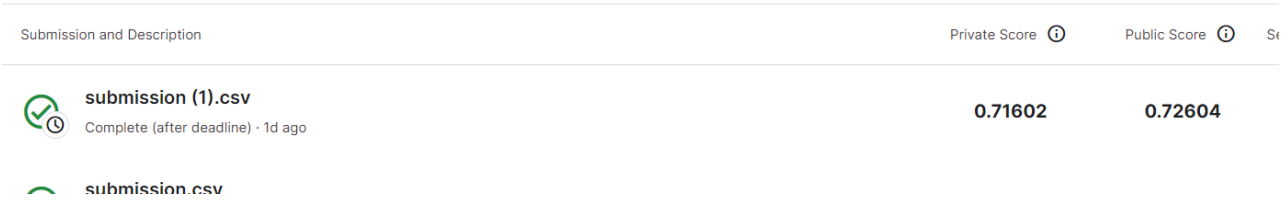


Figure 41: Achieved Kaggle Scores

17 Task to be tackled

17.0.0.1 In previous phase, the algorithms which have been used by us are decision tree classifier, random forest classifier, ada boost classifier, Multilevel perceptron classifier and XGB classifier. In phase 4 after getting the promising results in this phase, we planned on implementing the multi-layer perceptron model using PyTorch for this classification task which gave us average results. But this can be improved by tackling the skewed data for binary variables in our training data set. We observed that he model hardly gets its hand on training the examples on the features mapping to result 1 which makes it hard to predict on the held out test set. So this model won't be effective in the real world. The diagram below shows the concept of skewed data sets. The best way to tackle this is to perform log-transform on the dataset which will make the data curve more distributed. Another way would be to perform square root transform and Box-Cox transform.

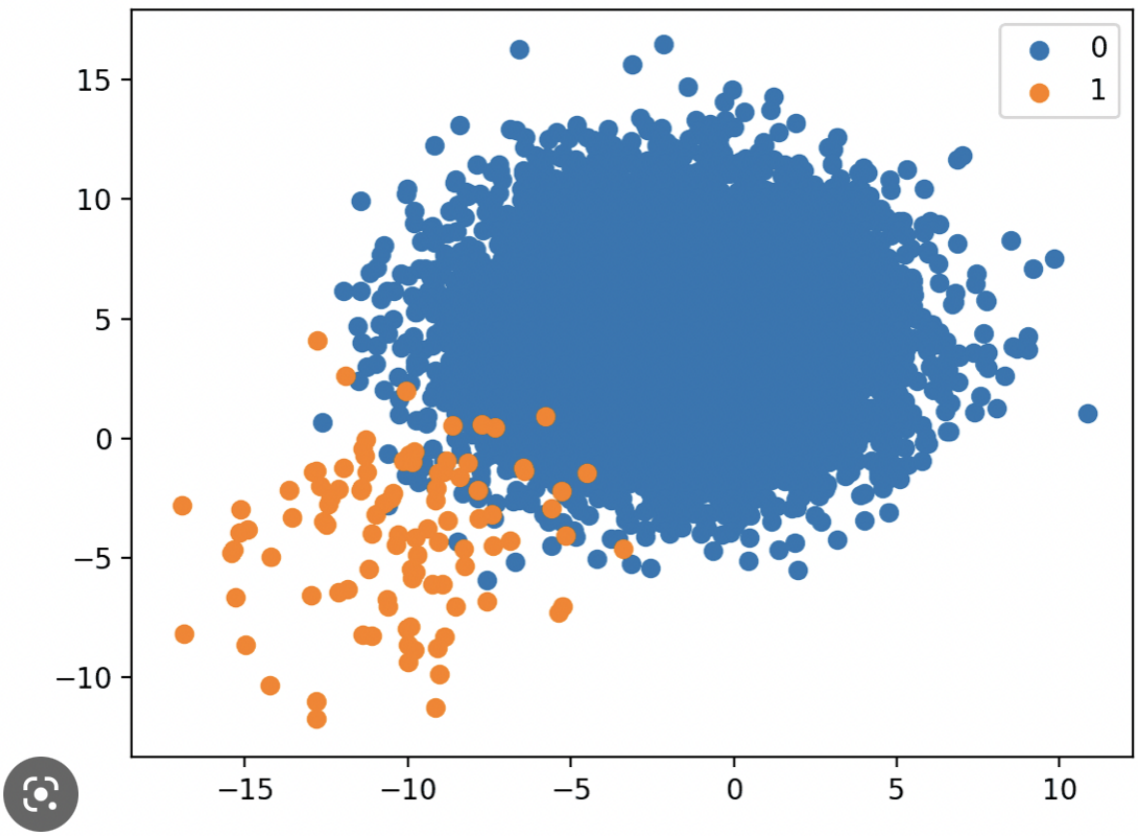


Figure 42: Skewed Data

18 Conclusion:

18.1 Project focus -

The holistic aim of this project is to predict which users are most likely to return the loan amount borrowed by them. The focus of this project in phase 3 was to perform feature engineering and hyperparameter tuning along with cross-validation. In Phase 4, we integrated PyTorch for the purpose of adding flexibility to training models for neural networks. Our primary focus on this phase was to build an Multi-Layered-Perceptron model for classification and regression with loss functions of Cross-Entropy and Mean-Squared-Error. We implemented both Classification and Regression using Pytorch with following metrics like MSE, CXE and

combination of MSE + CXE with the given achitecture models like Softmax, ReLu. The best model was Decision Tree algorithm which provided accuracy scale of around 0.92. Further, we got Kaggle scores with public score and private score being 0.726 and 0.716 respectively. For getting the kaggle scores, we computed metrics like Mean squared error, found accuracy and precision scale for which we understood how to go around.

18.2 Hypothesis

In this phase, we have implemented a multi layer perceptron model in order to perform classification as well as regression task on the HCDR dataset.

H0: Using Multi layer perceptron in pytorch, we can obtain a model which gives us heighest accuracy on the given dataset.

H1: Using Multi layer perceptron in pytorch, we are unable to obtain a model which gives us heighest accuracy on the given dataset.

18.3 Summary

- In order to get a better feel of the data, we computed the summary statistics and plotted histograms, displot, heatmaps, categorical plot, pair plot and countplot of the application train dataset.
- Correlation analysis was performed with respect to target columns and distribution of categorical and numerical variables with respect to target.
- Since the dataset consisted of many missing values, the next step was to perform missing value analysis.
- The rows with missing values were dropped and various imputer strategies were used in the numerical and categorical pipelines to handle the missing values
- Machine learning algorithms are then applied on these pipelines and accuracy scores are calculated and logged to display output in tabular format.
- By evaluating various machine learning algorithms, we came to the conclusion that logistic regression, decision tree and random forest algoithms perform the best on the held out test data set with higher accuracy in phase 2
- In phase 3, we performed feature engineering by deriving new features from available data sources like bureau and install payments that can add value to the model.
- The various pipelines which were created in phase 2 were then given as input to GrigSearchCV which predicted the optimum parameters as well as fit data on training set and predict target values for test set.
- In phase 4, we got to a conclusion that, our best solution was obtained in Architecture 3 using Relu for all hidden layers and Softmax for the output. In this combination, we only used Cross Entropy as our loss function with optimizer as Adam in our classification model. The particular configuration of 165-128-ReLu 128-128-ReLu 128-128-ReLu 128-128-ReLu 128-2-Softmax (Bias = True) yielded best output for the model.
- To improve the model future, we need to tackle the skewed nature of the data set.

18.4 Result Significance

Deriving features from other available data sources like Bureau, Installments Payment added value to the model. Our feature selection with our most significant features including EXT-SOURCE-1, EXT-SOURCE-2, EXT-SOURCE-3, DAYS-BIRTH and AMT-CREDIT our AUC Score improved to 0.7 on Kaggle (an increase of approximate 40 percent in the score). This proves that inclusion of Grid search with an optimal value of cross folds and feature selection improved the model performance overall. This can also be proved by the performance metrics of p-value which also dropped which insinuates the improved performance. For PyTorch metrics, there was one model which outperformed others in the pool of our created Architectures. The results of Architecture 3 suggested that amalgamation of 2 activation gave a slightly high performance compared to the others. The prominent model had configurations of Rectified Linear Unit with 165-128-ReLu 128-128-ReLu 128-128-ReLu 128-128-ReLu 128-2-Softmax (Bias = True). This configuration gave a loss metric of 0.394 which was the lowest amongst other models.

```
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=165, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=128, bias=True)
  (3): ReLU()
  (4): Linear(in_features=128, out_features=128, bias=True)
  (5): ReLU()
  (6): Linear(in_features=128, out_features=128, bias=True)
  (7): ReLU()
  (8): Linear(in_features=128, out_features=2, bias=True)
  (9): Softmax(dim=None)
)
--------------------------------------------------
Loss Function: CXE

Epoch 1: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.395       Valid Loss:0.394
Epoch 2: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 3: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 4: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 5: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 6: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 7: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 8: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 9: Train Accuracy: 0.919   valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 10: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 11: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 12: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 13: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 14: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 15: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 16: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 17: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 18: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 19: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
Epoch 20: Train Accuracy: 0.919  valid Accuracy: 0.92    Train Loss: 0.394       Valid Loss:0.394
--------------------------------------------------
0.5
```

Out[15]:

	Architecture string	Optimizer	Epochs	Train accuracy	Valid accuracy	Test accuracy	Test Loss
0	165-128-128-128-128-2	<class 'torch.optim.adam.Adam'>	20	91.9%	92.0%	91.9%	0.394

Figure 43: ReLu + Softmax Model (Best Model)

18.5 Future of project and closing thoughts

In phase 4 after getting the promising results in this phase, we planned on implementing the multi-layer perceptron model using PyTorch for this classification task which gave us average results. But this can be improved by tackling the skewed data for binary variables in our training data set. We observed that he model hardly gets its hand on training the examples on the features mapping to result 1 which makes it hard to predict on the held out test set. So this model won't be effective in the real world. The diagram below shows the concept of skewed data sets. The best way to tackle this is to perform log-transform on the dataset which will make the data curve more distributed. Another way would be to perform square root transform and Box-Cox transform.