POLITECNICO DI TORINO

ICT for Smart Societies

Master's Degree Thesis

Comorbidity prediction of depressive and cardiometabolic disorders using multitask learning

The Artificial Intelligence in Medicine lab @ Universitat de Barcelona

Candidate
MALIK ABBAS

Supervisors - UB
ANNA CASCARANO
PROF. KARIM LEKADIR

Supervisor - PoliTo
PROF. MONICA VISINTIN

March 2022

Summary

Cardiovascular diseases have a close link with diabetes and depression. Studies show that prevalence of diabetes and/or depression increases risk for cardiac events. Furthermore, there is growing evidence of bidirectional adverse interaction between diabetes and depression. It could be helpful to study comorbidity of these three diseases and predict in advance the occurrence of cardiovascular diseases in patients of diabetes and depression so that necessary intervention can be done in time to prevent any life-threatening event. Considering visible success of machine learning techniques on predictive tasks, it would be interesting to exploit them for prediction of aforementioned diseases. Data used for this thesis is obtained from UK Biobank which is a large-scale biomedical database and research resource, containing in-depth genetic and health information from half a million UK participants. The dataset comprises of a detailed information on lifestyle and medical conditions for each subject. For this thesis however, only exposome features have been considered which include lifestyle, sociodemographic factors, traumatic events and physical measures. Rational behind choosing these features is to predict risk of these life-threatening diseases without going through detailed and expensive medical examinations. Considering the objective of the thesis, a particular technique of machine learning called multitask learning was used. It solves multiple tasks at the same time while exploiting commonalities and differences among them. The proposed model is based on neural networks built using Keras library. This thesis would serve as a stepping stone in this regard, as to the best of our knowledge and research, a little work has been done that focuses on this problem along these lines. With availability of more data and improvements in the model in future, it could become an interesting tool to predict risks of diseases by filling an easily accessible online questionnaire.

Acknowledgements

I owe a lot to so many people for completing my thesis work. Without their continuous support and guidance, it would not be possible to learn and thoroughly enjoy this experience.

I would like to thank Prof. Karim Lekadir, director of BCN-AIM lab, for trusting in me and granting me an opportunity to be part of his team. I would also like to thank Prof. Monica Visintin from Politecnico di Torino for being my supervisor. Many thanks to Anna Cascarano, a PhD student in the lab, for mentoring and guiding me throughout the project. Her support and appreciation boosted my motivation and confidence at every step. I would also like to thank other lab members; Kaiser Kushibar, Esmeralda Ruiz Pujadas, Lidia Garrucho, Victor Campello, Vien Dang Ngoc, Carlos Martín Isla, Carla Sendra and Angélica Atehortua for sharing their knowledge and always being there when I needed their help. Working in their company certainly made the experience more delightful for me.

A special thanks to my late mother who blessed me and enabled me to be where I am today. Finally, thanks to all my friends and family for their encouragement and support.

Table of Contents

List of Tables

List of Figures

Acronyms

CVDs
Cardiovascular Diseases

ELS
Early Life Stress

UK
United Kingdom

ICD
International Classification of Diseases

MTL
Multitask Learning

PR
Precision Recall

ROC
Receiver Operating Characteristic

SMOTE-NC
Synthetic Minority Over-sampling Technique for Nominal and Continuous

TP
True Positive

FP
False Positive

TN

True Negative

FN

False Negative

Chapter 1

Introduction

1.1 Background and motivation

According to World Health Organization, about 422 million people worldwide have diabetes, the majority living in low-and middle-income countries. This metabolic disease characterized by elevated levels of blood glucose (or blood sugar) leads over time to serious damage to the heart, blood vessels, eyes, kidneys and nerves. Adults with diabetes historically have a two or three times higher rate of cardiovascular disease (CVD) than adults without diabetes [1].The risk of cardiovascular disease increases continuously with rising fasting plasma glucose levels, even before reaching levels sufficient for a diabetes diagnosis [2]. Depression is another common illness worldwide, with an estimated 3.8% of the population affected, including 5.0% among adults and 5.7% among adults older than 60 years. Approximately 280 million people in the world have depression [3]. There is strong evidence that depression is associated with an increased risk of cardiovascular disease and cardiac death [4]. In studies of large populations free of medical disease, after controlling for all known cardiovascular risk factors, it is those individuals with baseline depression that are more likely to develop cardiovascular disease [5]. However, there is not enough data that could help in evaluating prognostic implications of joint effects of diabetes and depression on cardiovascular disease and death.

Depression frequently occurs comorbidly with diabetes although it is unrecognized and untreated in approximately two thirds of patients with both conditions. The course of depression in patients with both diabetes and depression is chronic and severe. Up to 80% of patients with diabetes and depression will experience a relapse of depressive symptoms over a 5-year period [6]. Several studies have examined whether depression is a predictor for the onset of diabetes, most of which confirm an increase of risk of diabetes in depressed patients, but differ in the extent

of this increase that ranges between 32% and 60% [7, 8, 9, 10]. On the other hand, several studies assessed if diabetes may also increase the risk for depression. In the absence of comorbidities and complications, the data showed relationship between diabetes and the onset of depression [7, 8]; with an incidence estimate among 15% to 24% [7, 11, 12].

Unfortunately there is no adequate data that could help us evaluate the exact causal relationship among diabetes, depression and CVDs. However, there is a plenty of research that provides evidence of their coexistence and their adverse effects. In order to reduce risk factors for cardiovascular events among the patients, it is important to study and treat effective component of this disease combination.

1.2 Goals and contributions

This work emerges to address the need of a quantitative tool to predict the occurrence of a disease among diabetes, depression and CVDs given that a patient has already developed one or two other diseases from these three considered diseases. To this end, a machine learning technique called multitask learning was implemented. In multitask learning, multiple tasks are solved at the same time while exploiting the differences and commonalities across tasks. This generally results in improved learning efficiency and prediction accuracy compared to for task specific models compared to training the models separately. Since the underlying assumption in our case is that diabetes, depression and cardiovascular diseases have a correlation and underlying comorbidities, this technique becomes particularly interesting. Multitask learning performs better if the tasks have a correlation among them, otherwise using this technique could be counterproductive.

Another important objective of the thesis is to study the relationship of early life stress (ELS) on prevalence of diabetes, depression and CVDs. We know already how these disease co-occur and cause heavy societal and health burden. This work also explores the relationship of ELS with underlying mechanisms that cause co-occurrence of these diseases. For instance, ELS may trigger behavioural and biological mechanisms involved in the later development of depressive and cardiometabolic disorders and their comorbidity. In addition to features related to ELS, some other exposome features e.g. sociodemographic, lifestyle and physical measures have also been considered. Exposome features are a collection of environmental factors, such as stress and diet, to which an individual is exposed and which can have an effect on health.

To the best of our knowledge that focuses on using multitask learning and only

exposome features to predict occurrence of depressive and cardiometabolic disorders. This work is an effort to explore the possibility of predicting these diseases using easily accessible information and help physicians to see how likely it is for a patient to develop, for example, cardiovascular disease if they already have diabetes or/and depression.

1.3 Document structure

Chapter 1: Introduction to the problem and motivation of the thesis, including the specific goals and contributions.

Chapter 2: State of the art.

Chapter 3: Methods section explaining the dataset used, techniques implemented for data preprocessing, used multitask learning model and how the model has been implemented.

Chapter 4: Description of result metrics used for evaluation of model and detail of conducted experiments.

Chapter 5: A discussion summarising the challenges addressed and the obtained results, the limitations of the study and future perspectives. Finally the conclusions, with an overview of the main ideas.

Chapter 2

Related Work

There is little work done on using multitask learning to predict comorbidity of depressive and cardiometabolic disorders. However, there are some studies to investigate risk prediction for other multiple related diseases. One of them is the development of an optimization-based formulation that can simultaneously predict the risk for all diseases and learn the shared comorbidities [13]. Real Electronic Health Record database with patients at risk of Congestive Heart Failure and Chronic Obstructive Pulmonary Disease was used here to investigate the comorbidities of these diseases. Autism Spectrum Disorder (ASD) and Intellectual Disability (ID) are comorbid neurodevelopmental disorders with complex genetic architectures. Network-based gene risk prioritization algorithm named DeepND that performs cross-disorder analysis has been used to improve prediction power by exploiting the comorbidity of ASD and ID via multitask learning [14]. Another study in this area includes Multi-Task Learning for Mental Health using Social Media Text [15]. Initial groundwork for estimating suicide risk and mental health in a deep learning framework has been introduced here. By modeling multiple conditions, the system learns to make predictions about suicide risk and mental health at a low false positive rate. Conditions are modeled as tasks in a multi-task learning (MTL) framework, with gender prediction as an additional auxiliary task.

In summary, adapting multitask learning using exposome features to predict depression, diabtes and cardiovascular diseases remains a nontrivial task. Therefore, in this work, an MTL model has been developed specifically for these disease by exploiting only the exposome features.

Chapter 3

Methods

3.1 Dataset

The dataset used for the project is UK Biobank dataset which a large-scale biomedical database and research resource, containing in-depth genetic and health information from half a million UK participants. This was based on 1) Computed-based questionnaire on life-course exposures, medical history and treatments and 2) Self-reported diseases. While the dataset consists of numerous features including biomarkers and medical tests for various diseases, features that have been considered in this project are only exposome factors.These features can be categorized into following categories:

1. Lifestyle
2. Early life exposure
3. Physical measures
4. Sociodemographics
5. Traumatic events

Table 3.1 shows list of all considered features and their categories.

Table 3.1: List of features

No.	Feature Category	Feature Name	Feature Type
1	Lifestyle	sleepduration	Integer
2	Lifestyle	sleeplessness	Categorical
3	Lifestyle	currenttobacco	Categorical
4	Lifestyle	pasttobacco	Categorical
5	Lifestyle	cookedvegetable	Integer
6	Lifestyle	saladintake	Integer
7	Lifestyle	fruitintake	Integer
8	Lifestyle	driedfruit	Integer
9	Lifestyle	oilyfishintake	Categorical
10	Lifestyle	nonoilyfish	Categorical
11	Lifestyle	processedmeat	Categorical
12	Lifestyle	poultryintake	Categorical
13	Lifestyle	beefintake	Categorical
14	Lifestyle	lambintake	Categorical
15	Lifestyle	porkintake	Categorical
16	Lifestyle	cheeseintake	Categorical
17	Lifestyle	milkused	Categorical
18	Lifestyle	alcoholintake	Categorical
19	Lifestyle	breadintake	Integer
20	Lifestyle	breadtype	Categorical
21	Lifestyle	cerealintake	Integer
22	Lifestyle	cerealtype	Categorical
23	Lifestyle	smoking	Categorical
24	Lifestyle	salttofood	Categorical
25	Lifestyle	teaintake	Integer
26	Lifestyle	coffetype	Categorical
27	Lifestyle	alcoholdrinkerstatus	Categorical
28	Lifestyle	variationdiet	Categorical
29	Lifestyle	spreadtype	Categorical
30	Lifestyle	daytimedozing	Categorical
31	Lifestyle	napduringday	Categorical
32	Lifestyle	majordietarychang	Categorical
33	Lifestyle	summedMETminutes	Continuous
34	Lifestyle	waterintake	Integer
35	Lifestyle	nonbutterspread	Categorical
36	Lifestyle	amountalcoholdrunk	Categorical
37	Lifestyle	frequdrinkingalcohol	Categorical

Table 3.1: List of features

No.	Feature Category	Feature Name	Feature Type
38	Lifestyle	evertakencannabis	Categorical
39	Lifestyle	everbeeninjuredrinkalcohol	Categorical
40	Lifestyle	everhadknownpersonalcoh	Categorical
41	Lifestyle	freqdepressed	Categorical
42	Lifestyle	frequnenthusiasm2weeks	Categorical
43	Lifestyle	freqtensenness2weeks	Categorical
44	Lifestyle	freqtiredness2weeks	Categorical
45	Lifestyle	hardphysicalactivity	Categorical
46	Lifestyle	moderatephysicalactivity	Categorical
47	Lifestyle	softphysicalactivity	Categorical
48	Early life exposure	breastfedbaby	Categorical
49	Early life exposure	maternalsmoking	Categorical
50	Early life exposure	someonetakedoctorchild	Categorical
51	Early life exposure	sexuallymolestedchild	Categorical
52	Early life exposure	feltlovedchild	Categorical
53	Early life exposure	physicallyabusedfamiliychild	Categorical
54	Early life exposure	felthatedfamiliychild	Categorical
55	Early life exposure	adoptedasachild	Categorical
56	Physical measures	waistcircum	Continuous
57	Physical measures	hipcircum	Continuous
58	Physical measures	bmi	Continuous
59	Physical measures	Weight	Continuous
60	Physical measures	bodyfatpercent	Continuous
61	Physical measures	wholebodyfatmass	Continuous
62	Physical measures	wholebodyfatfreemass	Continuous
63	Sociodemographics	sex	Categorical
64	Sociodemographics	Age	Integer
65	Sociodemographics	qualification	Categorical
66	Sociodemographics	currentemployment	Categorical
67	Sociodemographics	ethnic	Categorical
68	Sociodemographics	agecompleted	Integer
69	Traumatic events	victimsexualassault	Categorical
70	Traumatic events	witnessedsuddenviolentdeath	Categorical
71	Traumatic events	victimphysicallyviolentcrime	Categorical
72	Traumatic events	beenseriousaccidentlifethrea	Categorical
73	Traumatic events	abletopaymortgageadult	Categorical
74	Traumatic events	sexualinterferenceadult	Categorical

Table 3.1: List of features

No.	Feature Category	Feature Name	Feature Type
75	Traumatic events	physicalviolenceadult	Categorical
76	Traumatic events	beenconfidingrelationadult	Categorical
77	Traumatic events	belittlementbypartneradult	Categorical
78	Traumatic events	avoidedactsstresspasthmonth	Categorical
79	Traumatic events	repeatedisturbingpastmonth	Categorical
80	Traumatic events	feltupsetstressfulpastmonth	Categorical
81	Traumatic events	diagnosedlifethreateningillnes	Categorical
82	Traumatic events	victimphysicallyviolentcrime	Categorical
83	Traumatic events	townsenddeprivation	Continuous

3.2 Data preprocessing

Before the dataset could be used for training the model, certain techniques were applied to make it clean and ready to use. This was done using Python and its Pandas library. These steps included:

1. The dataset loaded from csv to Pandas dataframe contained 4009 features out of which only the ones mentioned in table 3.1 were selected.

2. Each subject in the dataset contained many columns indicating diseases that were either diagnosed by a doctor or self reported by the subjects. Diabetes, depression and CVDs being the diseases of interest, columns regarding rest of the diseases were discarded. The dataset contained ICD 9 and 10 codes for the diagnosed diseases. The International Classification of Diseases (ICD) is a globally used diagnostic tool for epidemiology, health management and clinical purposes. The ICD provides a method of classifying diseases, injuries, and causes of death. The World Health Organization (WHO) publishes the ICDs to standardize the methods of recording and tracking instances of diagnosed disease all over the world, making it possible to conduct research on diseases, their causes, and their treatments. The ICD is currently the most widely used statistical classification system for diseases in the world. In addition, some countries—including Australia, Canada, and the United States—have developed their own adaptations of ICD, with more procedure codes for classification of operative or diagnostic procedures. The ICD has been revised periodically to incorporate changes in the medical field [16]. ICD-9 is 9th revision of ICD codes, published in 1975. Its implementation was formalized in 1979. ICD-9 codes consist of three to five digits with first character being numeric or alphabetic (E or V). In order to address the need of having a

flexible and expandable coding system, work on ICD-10 began in 1983. The latest version came into use in WHO Member States starting in 1994. One of ICD-9's issues is that some chapters are full and, thus were limited in the ability to add new codes. Because ICD-10 codes have increased in character length, the number of codes available for use has been greatly expanded. The biggest difference between the two code structures is that ICD-9 had 14,4000 codes, while ICD-10 contains over 69,823. ICD-10 codes consists of three to seven characters with first character being alphabetic.

In the considered dataset, diseases were reported in the form of ICD-9 and ICD-10 codes, hence there was a need to consolidate this information. In order to obtain a single column for each disease, multiple columns for each subject were checked to confirm presence of a disease. ICD9 and ICD10 codes included in diabetes, depression and CVDs are listed in tables 3.1-3.6. Final columns for each disease would be binary classified where 1 stands for presence of disease and 0 for absence of disease.

Code	Disease description
250	Diabetes mellitus

Table 3.2: ICD9 code included in diabetes

Code	Disease description
E08-E13	Diabetes mellitus

Table 3.3: ICD10 code included in diabetes

3. The dataframe obtained after consolidating disease columns now contains three binary columns each representing diabetes, depression and CVDs. Total rows in the dataframe are 501,948. There are 23 columns for which data is available only for 157,201 rows. This is because the questionnaire related to these columns was sent only to some of the participants. In order not to lose information, the dataset was split into 2 datasets, as depicted in figure 3.1. Dataset A contains 60 columns and 501948 rows whereas dataset B contains 23 columns and 157,287 rows. These datasets will be used together to train the model.

4. In column named *currentemployment*, there are 8 categories which represent different groups as shown in table 3.8. Some of the categories could be merged to obtain fewer number of categories, as shown in table 3.9. Similarly, feature called *ethnic* has been regrouped from original, as in table 3.10, to new groups

Code	Disease description
290-294	Organic Psychotic Conditions
295-299	Other Psychoses
300-316	Neurotic Disorders, Personality Disorders, And Other Nonpsychotic Mental Disorders
317-319	Intellectual Disabilities

Table 3.4: ICD9 codes included in depression

Code	Disease description
F30-F39	Mood [affective] disorders

Table 3.5: ICD10 codes included in depression

as shown in table 3.11. As both these columns i.e. *currentemployment* and *ethnic* contain categorical values. It is important to convert these label values into numeric values for our model to operate on them. To this end, one-hot encoding has been applied to integer representation of these columns. As a result of this, previously applied integer encoding from these columns is removed and a new binary variable is added for each unique integer value.

5. In both the newly created datasets i.e. dataset A and dataset B, there are still some missing values which need to be imputed. Imputation has been performed by using its mode if the feature is categorical or integer and its mean if the feature is continuous.

3.3 Network architecture

As discussed earlier, objective of the project is to use multitask learning to predict diabetes, depression and CVDs. Instead of developing a separate model for each task (or a disease in this case), a single model using MTL was developed that was jointly trained for all tasks. In this work, MTL was implemented in deep neural network using Keras. The model essentially has some common hidden layers for all tasks as well as some task specific layers, an approach called hard parameter sharing. This is displayed in figure 3.2
Functional API method in Keras library was used to build the model. Number of layers and units in each layer were selected based on manual search method. Different values for for the hypermeters were tried over a range of values and model hypermeters corresponding to best model performance were chosen. RelU activation was used in each hidden layer whereas sigmoid activation function was used in output layers. In order to ensure regularization and hence avoid overfitting,

Code	Disease description
390-392	Acute Rheumatic Fever
393-398	Chronic Rheumatic Heart Disease
410-414	Ischemic Heart Disease
415-417	Diseases Of Pulmonary Circulation
420-429	Other Forms Of Heart Disease
430-438	Cerebrovascular Disease
440-449	Diseases Of Arteries, Arterioles, And Capillaries
451-459	Diseases Of Veins And Lymphatics, And Other Diseases Of Circulatory System

Table 3.6: ICD9 codes included in CVDs

Code	Disease description
I00-I02	Acute Rheumatic Fever
I05-I09	Chronic Rheumatic Heart Disease
I20-I25	Ischemic Heart Disease
I26-I28	Pulmonary Heart Disease and Diseases of Pulmonary Circulation
I30-I5A	Other Forms Of Heart Disease
I60-I69	Cerebrovascular Disease
I70-I79	Diseases Of Arteries, Arterioles, And Capillaries
I80-I89	Diseases of veins, lymphatic vessels and lymph nodes, not elsewhere classified
I95-I99	Other and unspecified disorders of the circulatory system

Table 3.7: ICD10 codes included in CVDs

dropout was also introduced. As discussed in chapter 4, different experiments were performed where different number of output tasks were considered. Figure 4.3 shows architecture for experiment 1 where all the 3 diseases have been considered.

3.4 Implementation details

Keras, an open source software library that provides a Python interface for artificial neural networks, was used to implement the MTL model. Key implementation steps that were performed on all experiments are:

1. 2 percent of the dataset was separated from the main dataset. This holdout test dataset will be used to evaluate the performance of model. Remaining

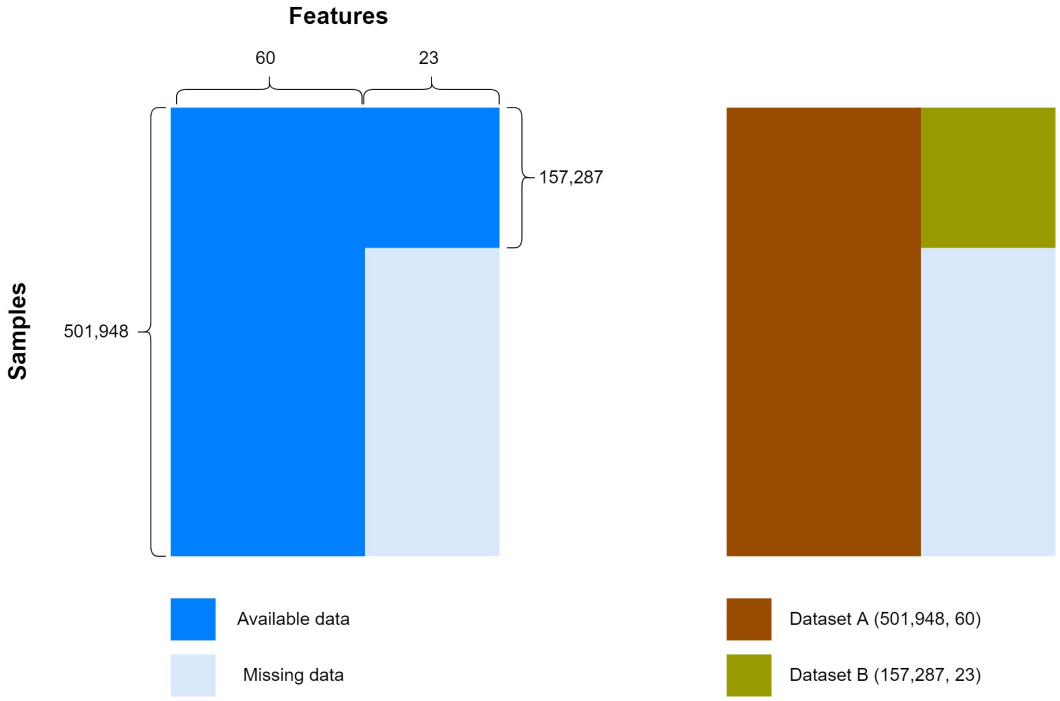


Figure 3.1: Dataset distribution. left: Available and missing data, right: Split of dataset

Category	Detail
1	In paid employment or self-employed
2	Retired
3	Looking after home and/or family
4	Unable to work because of sickness or disability
5	Unemployed
6	Doing unpaid or voluntary work
7	Full or part-time student
8	None of the above

Table 3.8: *currentemployment* feature - original categories

98% of the data was used to train the model with 5-fold cross validation.

2. Since values in the dataset have different ranges, they were scaled using normalization method. This step is necessary because without normalization gradients may end up taking a long time and can oscillate back and forth and take a long time before it can finally find its way to the global/local minimum.

Category	Detail
0	Unemployed & Full or part-time student
1	Looking after home and/or family & Doing unpaid or voluntary work & Unable to work because of sickness or disability & other
2	Retired
3	In paid employment or self-employed

Table 3.9: *currentemployment* feature - new categories

Category	Detail
1	White
1001	British
2001	White and Black Caribbean
3001	Indian
4001	Caribbean
2	Mixed
1002	Irish
2002	White and Black African
3002	Pakistani
4002	African
3	Asian or Asian British
1003	Any other white background
2003	White and Asian
3003	Bangladeshi
4003	Any other Black background
4	Black or Black British
2004	Any other mixed background
3004	Any other Asian background
5	Chinese
6	Other ethnic group
-1	Do not know
-3	Prefer not to answer

Table 3.10: *ethnic* feature - original categories

3. Batch size of 500, number of epochs as 5, loss function for each task as binary cross-entropy, adam optimizer with learning rate as 1×10^{-3} was used. These values were selected based on random trial of parameters over different values.

Category	Detail
1	White
2	Mixed
3	Asian or Asian British
4	Black or Black British
5	Other

Table 3.11: *ethnic* feature - new categories

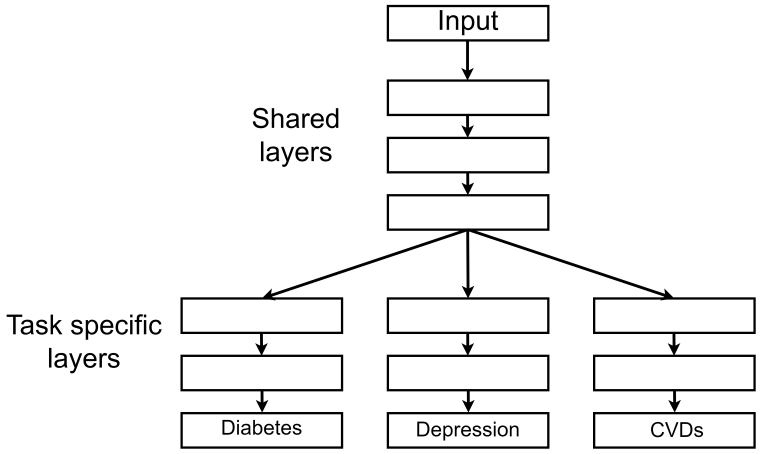


Figure 3.2: Hard parameter sharing for multi-task learning in deep neural networks

4. Combined number of features in dataset A and B are 90. In order to eliminate less important features and reduce training time, SelectKBest feature selection was performed. SelectKBest is a Scikit-learn API's class to extract best features from a dataset. This method selects features according to the k highest score based on some metric. Sklearn provides some predefined score functions which can be used as a metric. In this work, best performance was obtained when **k** was set to 60 and `score_func` to *f_classif*.

5. As shown in figure 3.3, the dataset is imbalanced particularly in case of diabetes. This is true for both, dataset A and dataset B. Before the data is fed to neural network, it needs to be balanced. Also, number of samples in both datasets need to be equal. To address this problem, first dataset A was balanced using SMOTE-NC technique. Synthetic Minority Over-sampling Technique for Nominal and Continuous or SMOTE-NC for short is an approach used to synthesize new examples in the dataset. In this technique, a random example from the minority class is first chosen. Then k of the nearest neighbors for that example are found (k=3 in this work). A randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point

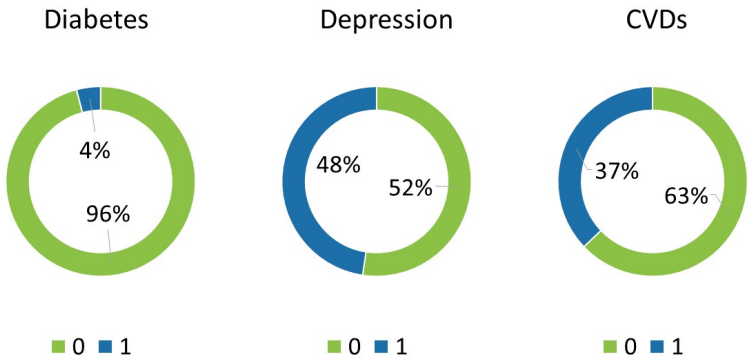


Figure 3.3: Data distribution according to the target variable

between the two examples in feature space.

After oversampling dataset A with SMOTE-NC, dataset B was oversampled using random oversampling technique. In this approach, subjects were randomly duplicated so that number of samples in dataset B matched those in dataset A.

For both, training and test datasets, rows in dataset A and B corresponding to same target class are concatenated and fed to the model. In this model, there are 3 tasks in experiment 1 and 2 tasks in experiments 2, 3 and 4 (described in chapter 4). In all experiments, binary crossentropy loss functions were used because each task performs binary classification. The model optimizes weighted sum of individual tasks.

6. Finally, hold out test data was used on the trained model to evaluate its performance.

Chapter 4

Results

4.1 Performance evaluation

Each task trained in the MTL model is actually a binary classification problem. Given an information about the predictors (features), the model has to predict whether the subject has a disease i.e. 1 for having a disease and 0 for no disease. Depending upon the experiment, there are a number of tasks in the MTL model. Each branch in the model represents a task that independently predicts the occurrence of a disease based on the input information.

Sigmoid activation function used in the output layer of each task predicts the probability against each input. In order to convert the probability to class 1 or 0, an appropriate threshold is needed. In this case, a probability threshold was chosen that maximized the f-score. F-score (or F_1 score) is given by:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{4.1}$$

In order to obtain the best threshold, a technique called threshold moving was used on precision recall curve. Threshold moving is a technique used to convert a predicted probability or scoring into a class label. The default value for the threshold is 0.5 for normalized predicted probabilities. However, default threshold did not represent an optimal interpretation of the predicted probabilities because of class imbalance in the dataset. In the process of threshold moving, first the model is fit on a training dataset and predictions are made on a test dataset. The predictions are in the form of normalized probabilities or scores that are transformed into normalized probabilities. Different threshold values are then tried and the resulting labels are evaluated using a chosen evaluation metric (f-score in this case). The threshold that achieves the best evaluation metric is then adopted for the model. After achieving the best threshold, predicted probabilities for the test set were

converted to class labels using the best threshold and rest of the metrics were calculated again. Besides f-score, other metrics includes ROC curve and confusion matrix. An ROC curve is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate
- False Positive Rate

True Positive Rate (TPR) is a synonym for recall and is therefore defined as follows:

$$TPR = \frac{TP}{TP + FN} \tag{4.2}$$

False Positive Rate (FPR) is defined as follows:

$$FPR = \frac{FP}{FP + FN} \tag{4.3}$$

A confusion matrix was also computed to understand further the prediction results. It gives an insight not only into the errors being made by the classifier but more importantly the types of errors that are being made. Confusion matrix uses four kinds of results; true positives, false negatives, false positives, and true negatives along with the positive and negative classifications. The four outcomes are then formulated in a 2×2 confusion matrix (for binary classification), as shown in table 4.1.

		Predicted	
	Total population	**Positive (PP)**	**Negative (PN)**
True	**Positive (P)**	**True positive (TP)**	**False negative (FN)**
	Negative (N)	**False positive (FP)**	**True negative (TN)**

Table 4.1: A 2x2 confusion matrix

4.2 Experiment 1: Subjects with diabetes, depression and CVDs

This experiment was conducted using the entire dataset where subjects might have none, one, two or all of the diseases among diabetes, depression and CVDs. Figure 4.2 displays best 60 features selected by k best method for this experiment. They have plotted in decreasing order of importance from top to bottom. *Waist circumference* is most important feature followed by *age*. Figure 4.3 shows the MTL model used for experiment 1. Since all 3 diseases have been considered for this experiment, it contains 3 branches. Table 4.2 shows summary of the model. Input size refers to the training size after performing over sampling whereas output size refers to output size for all 3 tasks.

Input size	**3,622,912 x 60**
Output size	**3,622,912 x 3 x 1**
Number of layers	**24**
Total params	**26,118**
Trainable params	**26,118**
Non-trainable params	**0**

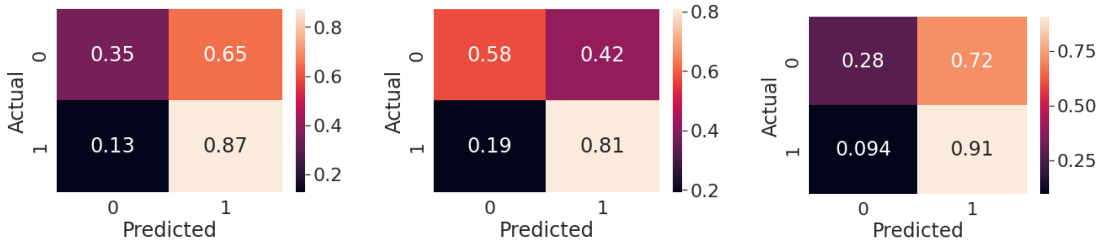
Table 4.2: Model summary for experiment 1

Figures 4.4, 4.5, 4.6 show Receiver operating characteristic and precision-recall curves for diabetes, depression and CVDs respectively. Black dot on these curves corresponds to probability threshold that leads to best f-score. Area under the curve of these plots corresponds to classification accuracy of the model. ROC Curves summarize the trade-off between the true positive rate and false positive rate for the model using different probability thresholds. On the other hand, PR curves summarize the trade-off between the true positive rate and the positive predictive value for the model using different probability thresholds. PR curves are more important in this case because objective is to find best compromise between precision and recall. In other words, results for the positive class (i.e. 1) are more important for us. It is useful to filter subjects with the disease through the model and then verify the diagnosis through further medical examinations. In this case, having more false positives is considered to be better than having more false negatives to avoid the patients being overlooked for needed precautions or treatment. The plots in these figures show that model performs best in case of depression.

Using best probability threshold obtained through threshold moving on AUC-PR curve, classification results for each disease were obtained on test data. Table 4.3 contains best performance metrics for all considered diseases and figure 4.1 shows confusion matrices for each disease.

	Precision	Sensitivity	Specificity	Accuracy	PR AUC	F-score
Diabetes	0.574	0.874	0.351	0.612	0.707	0.693
Depression	0.656	0.808	0.578	0.693	0.774	0.724
CVD	0.558	0.906	0.283	0.594	0.678	0.691

Table 4.3: Best performance metrics for experiment 1



(a) Diabetes **(b)** Depression **(c)** CVDs

Figure 4.1: Confusion matrices for experiment 1

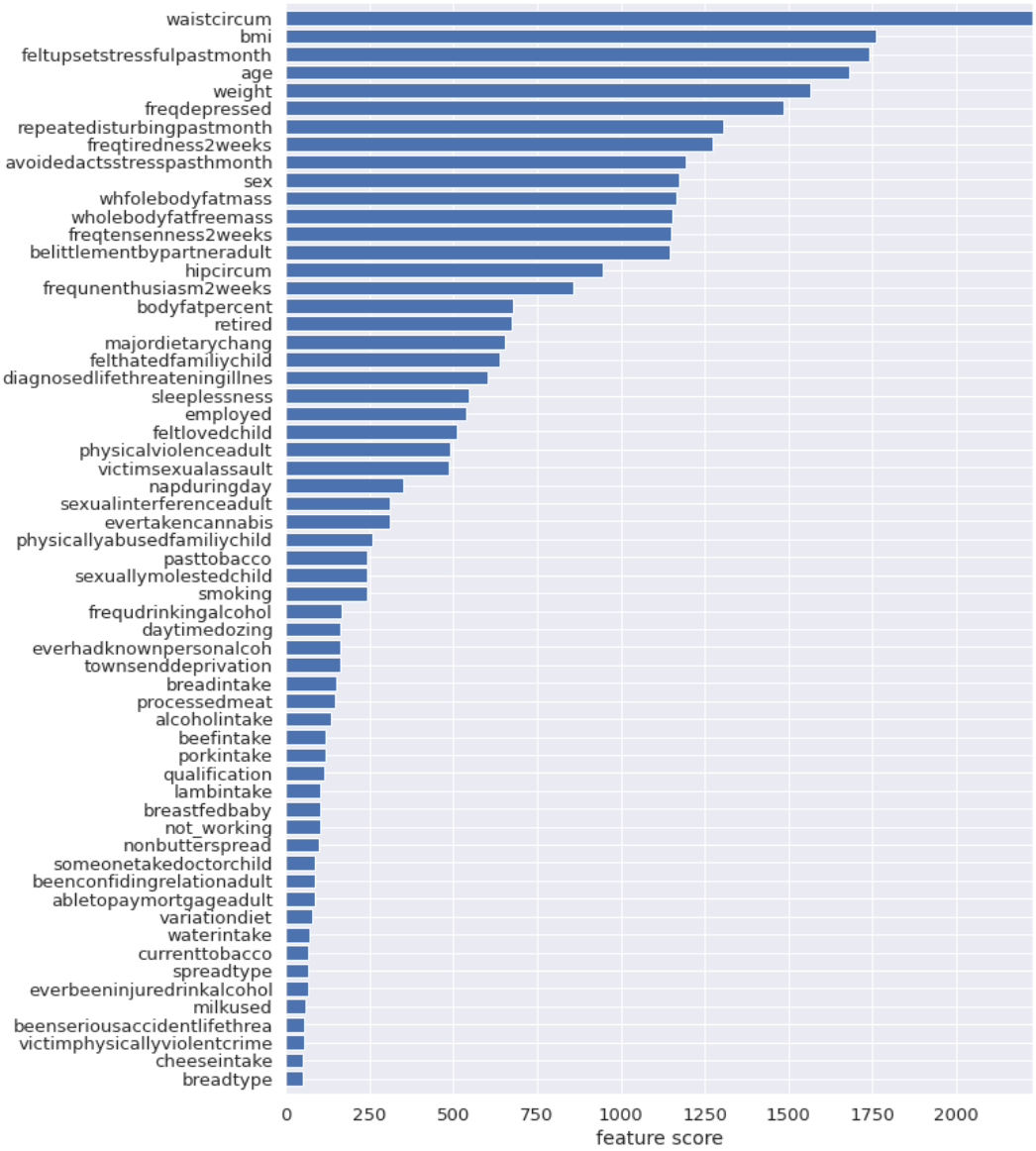


Figure 4.2: Feature selection results for experiment 1

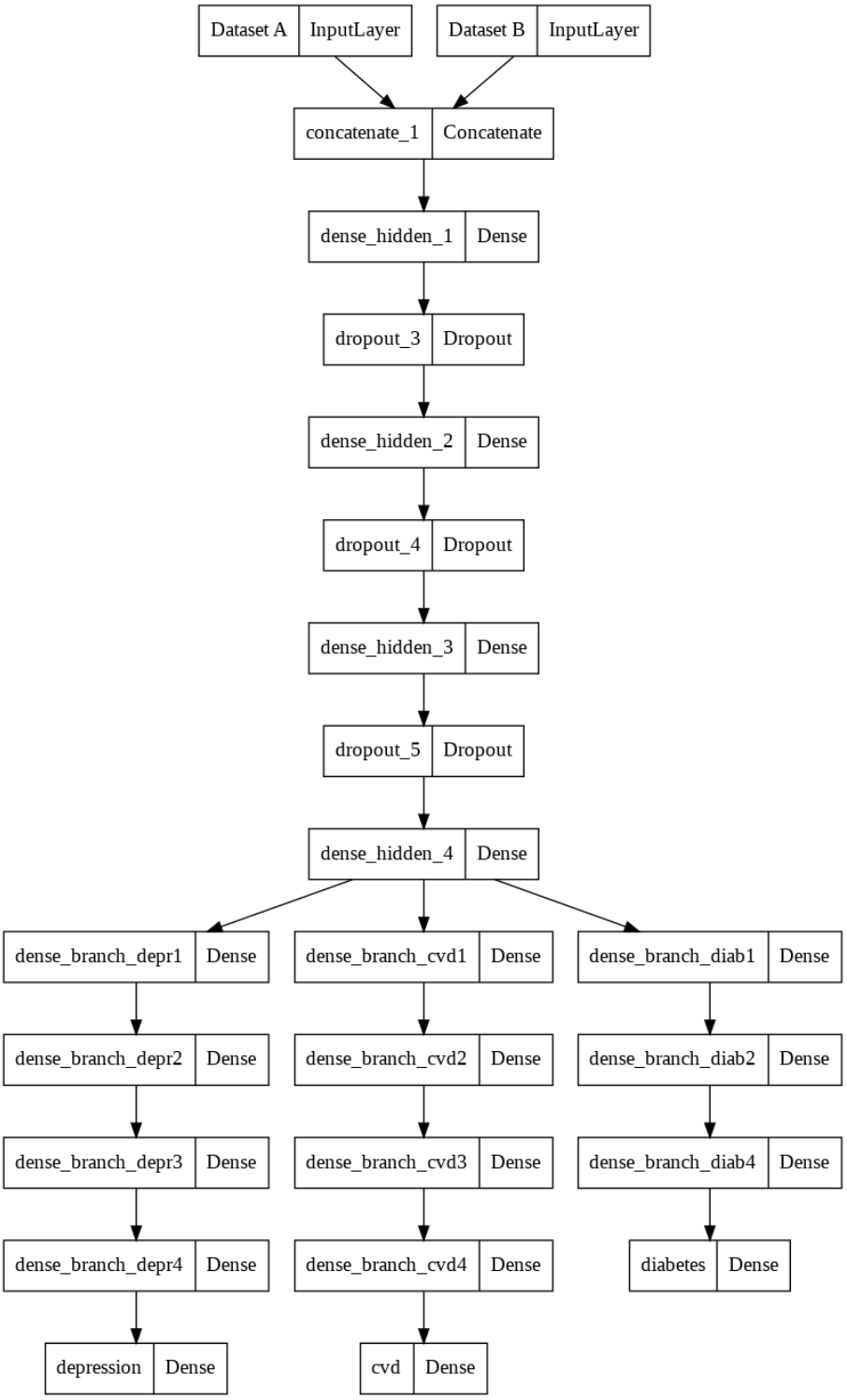
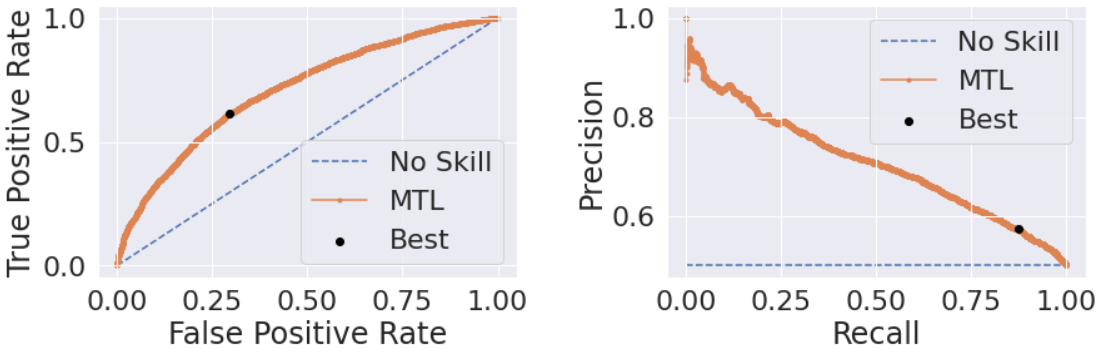
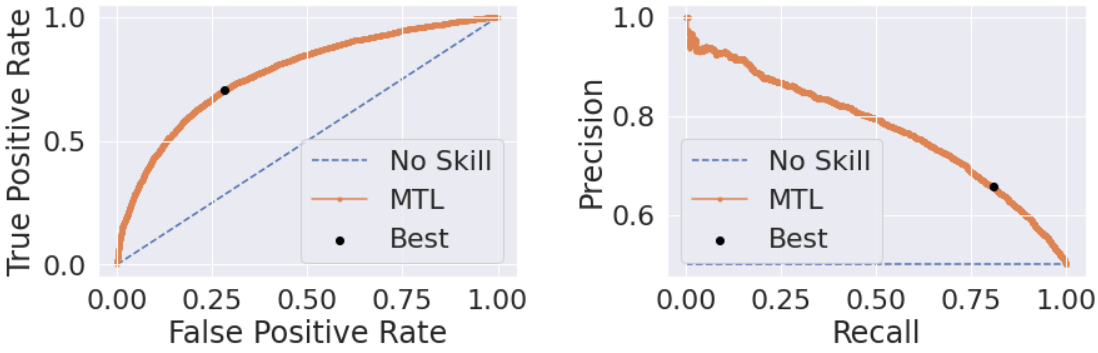


Figure 4.3: MTL model for experiment 1



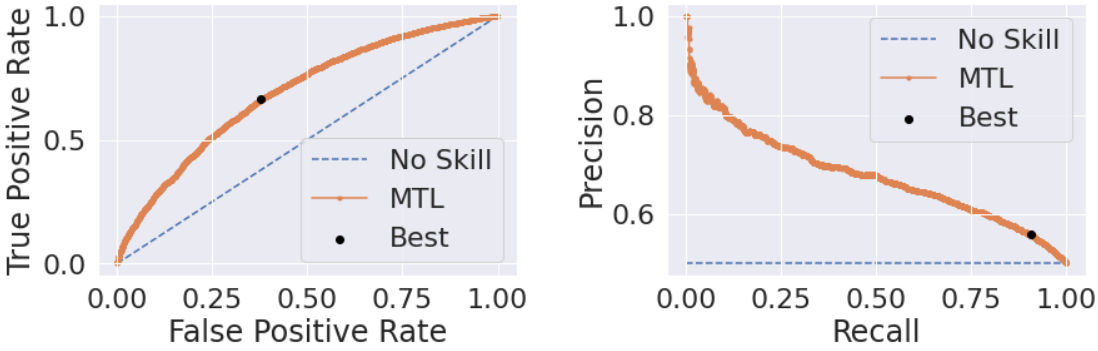
(a) ROC curve **(b)** PR curve

Figure 4.4: ROC and PR curves for diabetes



(a) ROC curve **(b)** PR curve

Figure 4.5: ROC and PR curves for depression



(a) ROC curve **(b)** PR curve

Figure 4.6: ROC and PR curves for CVDs

4.3 Experiment 2: Subjects with diabetes and depression

This experiment was carried out using subjects in dataset with none, one or both of the diseases among diabetes and depression. Results of this model would give an insight on how these two diseases are similar (or dissimilar). Figure 4.10 displays best 60 features selected by k best method for this experiment. As in experiment 1, *waist circumference* is still the most important feature for prediction of these diseases. Selected features and their corresponding importance scores in this experiment are different than those in experiment 1 because of different number of target diseases. Figure 4.11 shows the MTL model used for experiment 1. Since, only 2 diseases have been considered for this experiment, it contains 2 branches. Table 4.4 shows summary of the model.

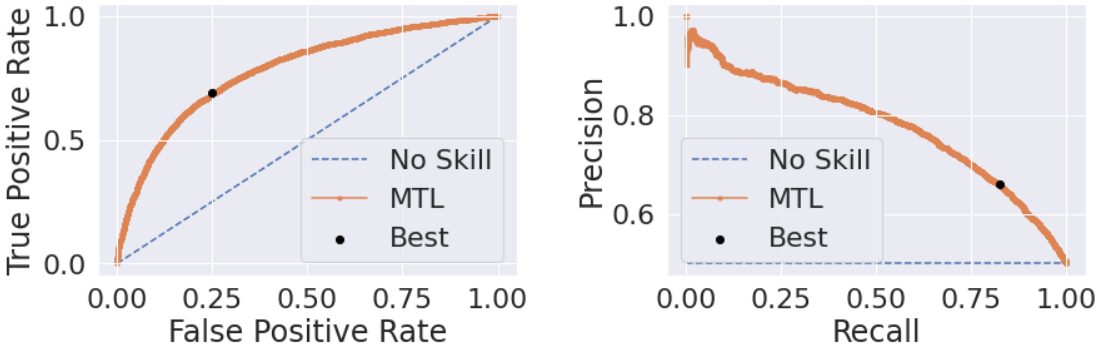
Input size	**2,932,704 x 60**
Output size	**2,932,704 x 2 x 1**
Number of layers	**19**
Total params	**17,277**
Trainable params	**17,277**
Non-trainable params	**0**

Table 4.4: Model summary for experiment 2

Figures 4.7 and 4.8 show Receiver operating characteristic and precision-recall curves for diabetes and depression respectively. Black dot on these curves corresponds to probability threshold that leads to best f-score. Using best probability score obtained through threshold moving on AUC-PR curve, classification results for each disease were obtained on test data. Table 4.5 contains best performance metrics for all considered diseases. Figure 4.9 shows confusion matrices for both diseases in this experiment.

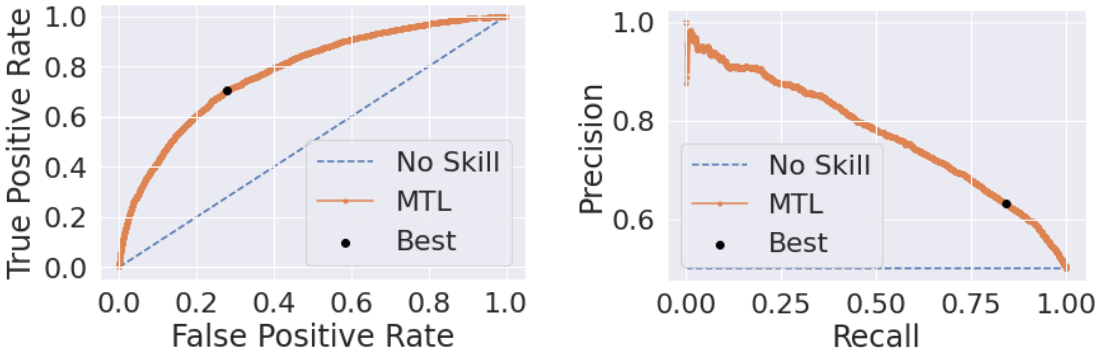
	Precision	Sensitivity	Specificity	Accuracy	PR AUC	F-score
Diabetes	0.659	0.825	0.573	0.699	0.779	0.732
Depression	0.633	0.839	0.514	0.677	0.773	0.722

Table 4.5: Best performance metrics for experiment 2



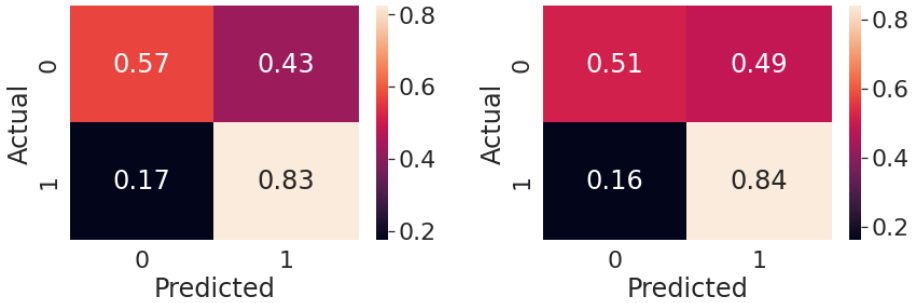
(a) ROC curve **(b)** PR curve

Figure 4.7: ROC and PR curves for diabetes



(a) ROC curve **(b)** PR curve

Figure 4.8: ROC and PR curves for depression



(a) Diabetes **(b)** Depression

Figure 4.9: Confusion matrices for experiment 2

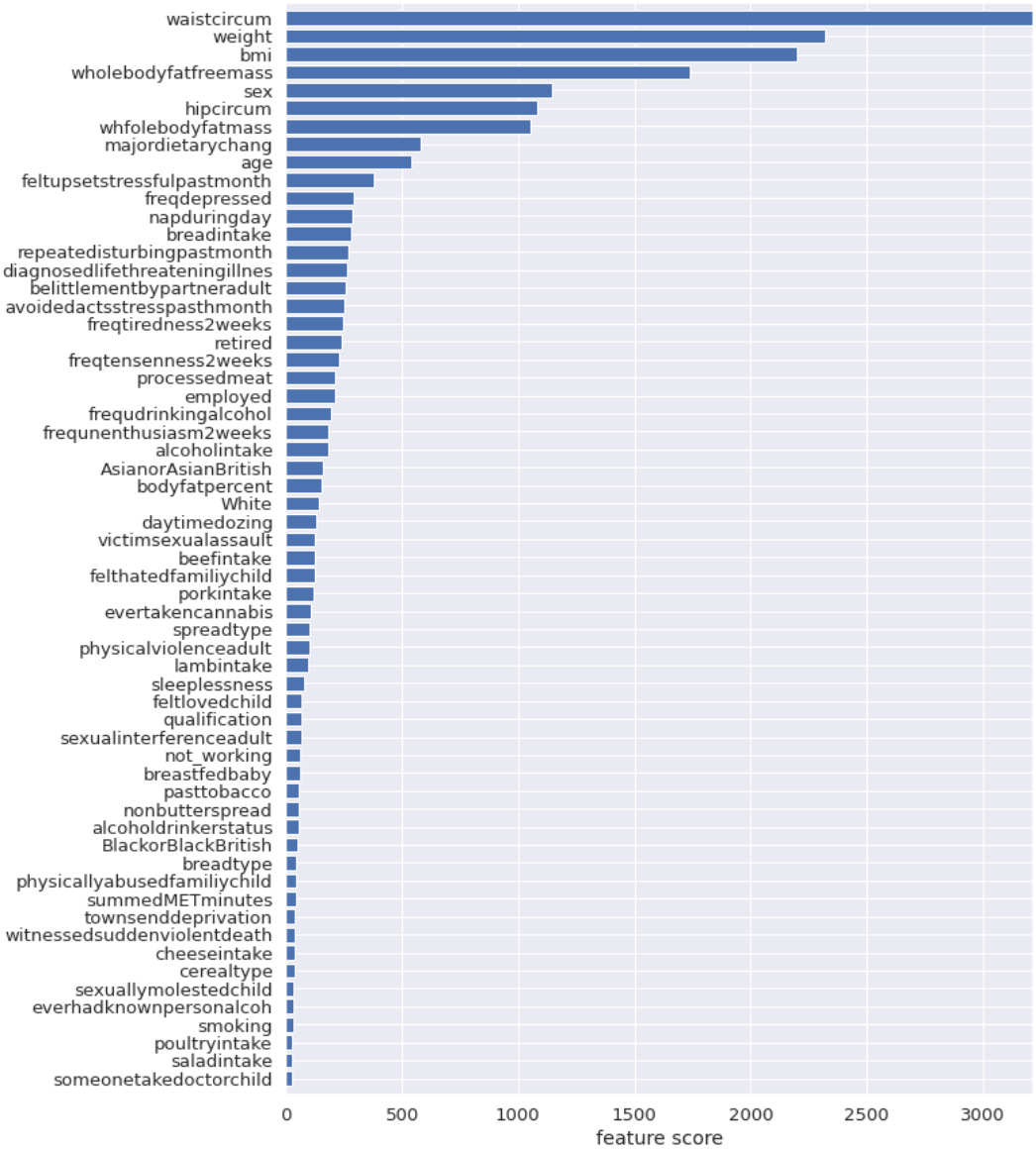


Figure 4.10: Feature selection results for experiment 2

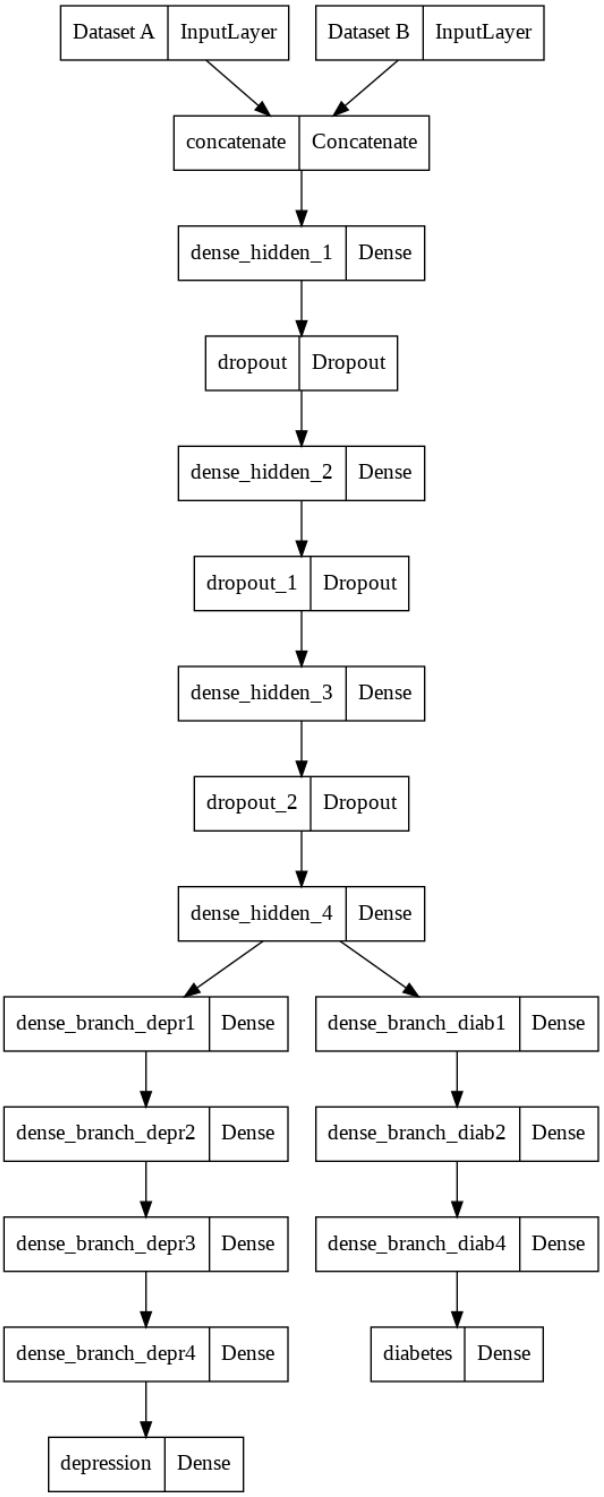


Figure 4.11: MTL model for experiment 2

4.4 Experiment 3: Subjects with diabetes and CVDs

This experiment was conducted using subjects in dataset with none, one or both of the diseases among diabetes and CVDs. As expected, results of this experiment were different than previous experiments because of a different set of data and a variation in feature similarity. Figure 4.15 displays best 60 features selected by k best method for this experiment. As expected, selected features and their scores are different in this experiment compared to those in previous experiments. Figure 4.16 shows the MTL model used for experiment 1. Since only 2 diseases were considered for this experiment, it contains 2 branches. Table 4.6 presents summary of the model.

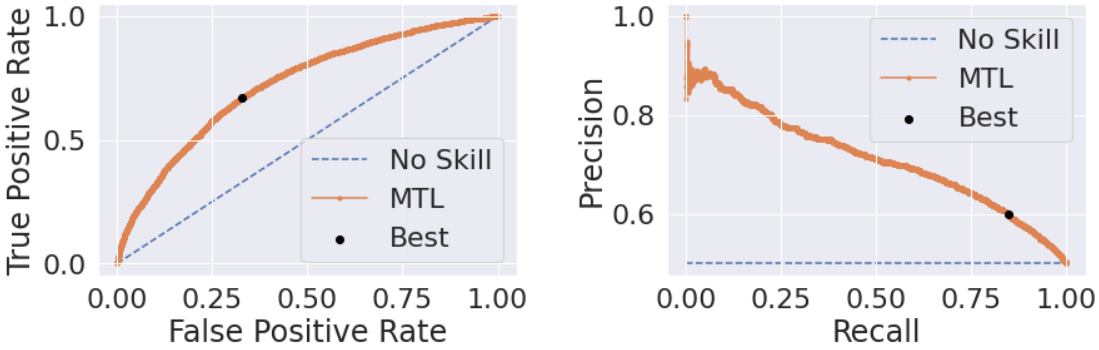
Input size	**3,067,408 x 60**
Output size	**3,067,408 x 2 x 1**
Number of layers	**16**
Total params	**15,382**
Trainable params	**15,382**
Non-trainable params	**0**

Table 4.6: Model summary for experiment 3

Figures 4.12 and 4.13 show Receiver operating characteristic and precision-recall curves for diabetes and CVDs respectively. AUC-PR for diabetes is higher than that for CVD. Using the best probability threshold obtained through threshold moving on PR curve, rest of the performance metrics were computed again which are listed in table 4.7. Figure 4.14 shows confusion matrices for diabetes and CVD. The model classifies TN and TP with 67% accuracy which is a really good improvement compared to experiment when all diseases are considered together.

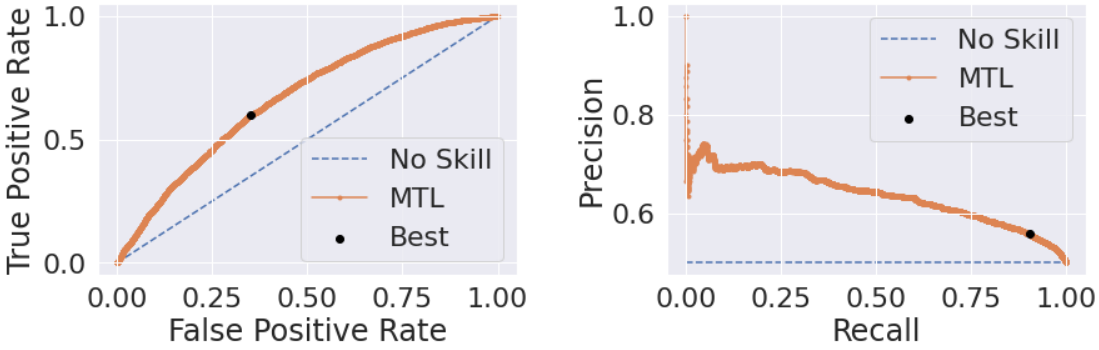
	Precision	Sensitivity	Specificity	Accuracy	PR AUC	F-score
Diabetes	0.67	0.67	0.67	0.67	0.713	0.67
CVDs	0.631	0.6	0.65	0.625	0.637	0.615

Table 4.7: Best performance metrics for experiment 3



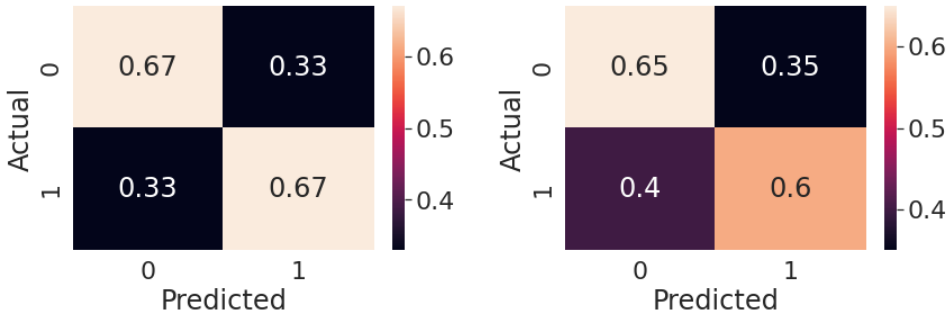
(a) ROC curve **(b)** PR curve

Figure 4.12: ROC and PR curves for diabetes



(a) ROC curve **(b)** PR curve

Figure 4.13: ROC and PR curves for CVDs



(a) Diabetes **(b)** CVD

Figure 4.14: Confusion matrices for experiment 3

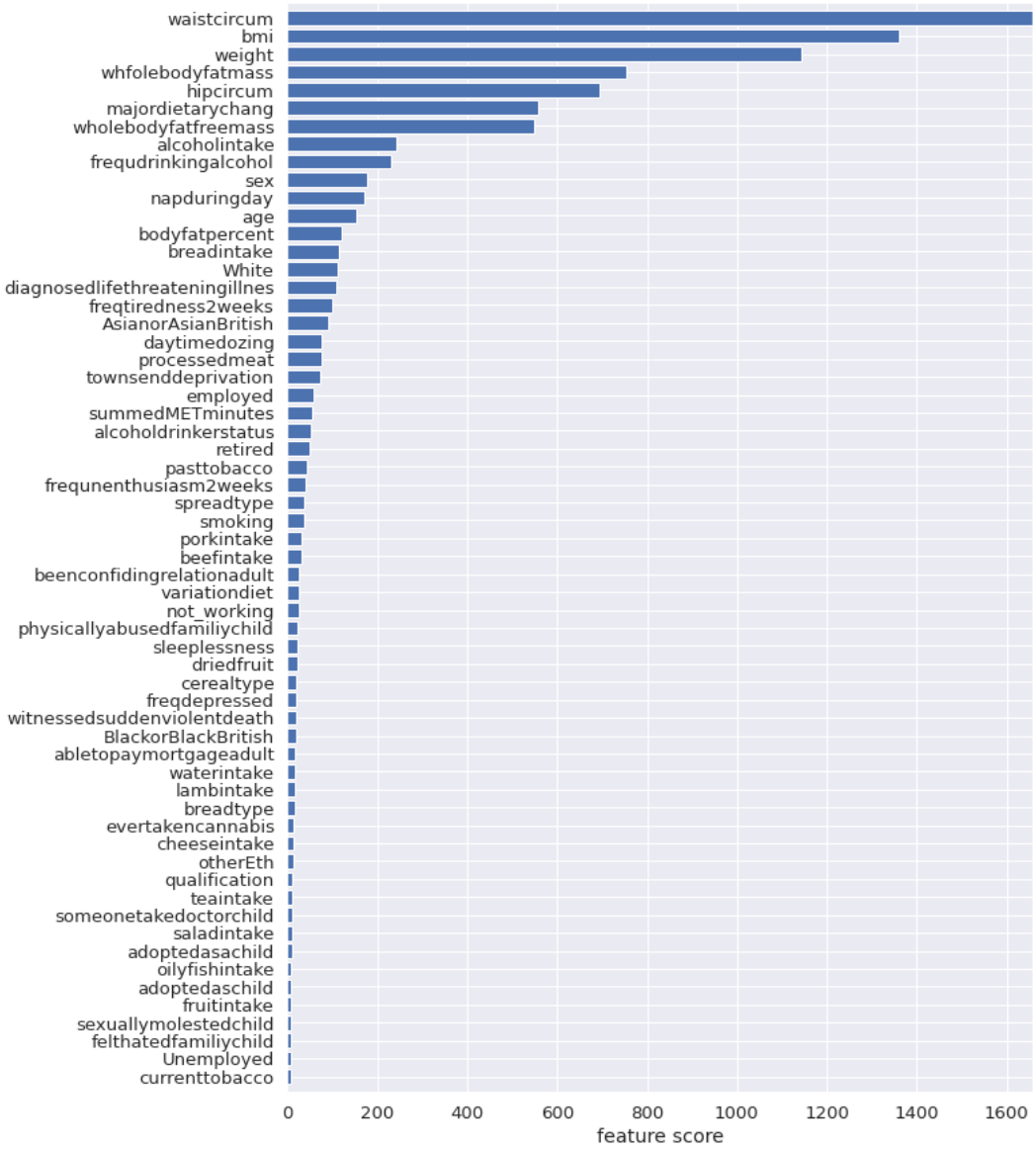


Figure 4.15: Feature selection results for experiment 3

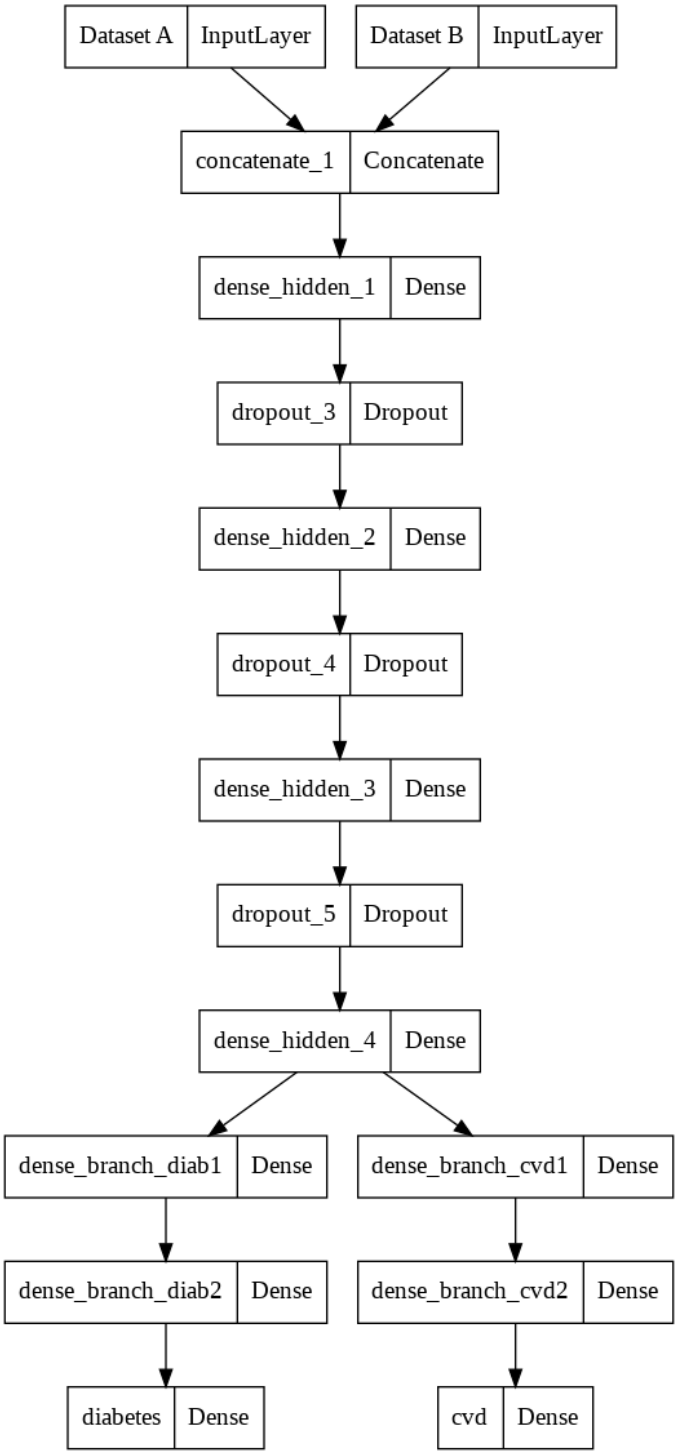


Figure 4.16: MTL model for experiment 3

4.5 Experiment 4: Subjects with depression and CVDs

This experiment was conducted using subjects in dataset with none, one or both of the diseases among depression and CVDs. Figure 4.20 displays best 60 best features selected by k best method for this experiment. As opposed to previous experiments, *age* is the most important feature followed by feature *waist circumference*. Figure 4.21 shows the MTL model used for this experiment. This model also contains 2 branches because number of considered tasks (i.e. diseases) is 2. Model is summary is given in table 4.8.

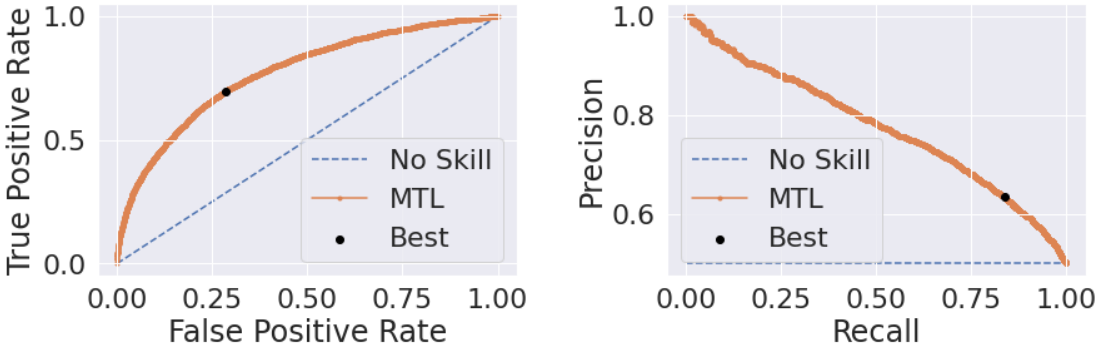
Input size	**1,999,392 x 60**
Output size	**1,999,392 x 2 x 1**
Number of layers	**20**
Total params	**17,952**
Trainable params	**17,952**
Non-trainable params	**0**

Table 4.8: Model summary for experiment 4

Figures 4.17 and 4.18 show Receiver operating characteristic and precision-recall curves for depression and CVDs respectively. Black dot on these curves corresponds to probability threshold obtained through threshold moving on PR curve. Using this chosen threshold, classification results for each disease were computed on test data. Table 4.9 contains best performance metrics for all considered diseases. Figure 4.19 shows confusion matrices for both considered diseases in this experiment.

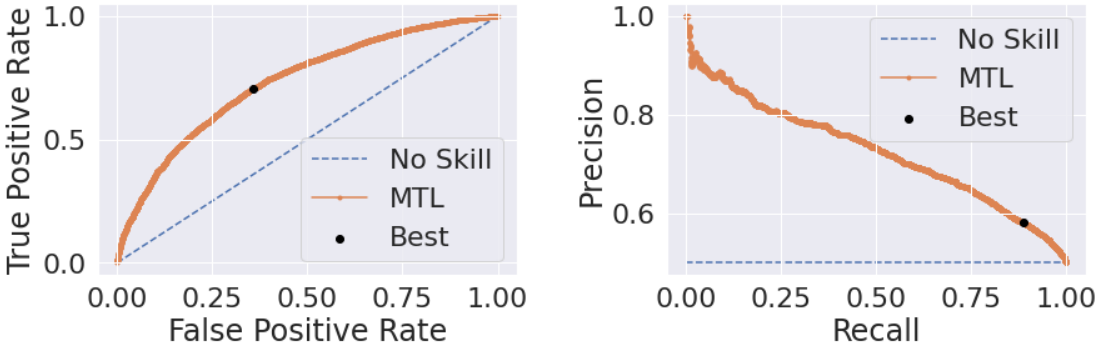
	Precision	Sensitivity	Specificity	Accuracy	PR AUC	F-score
Depression	0.634	0.837	0.516	0.677	0.776	0.721
CVDs	0.582	0.885	0.365	0.625	0.726	0.702

Table 4.9: Best performance metrics for experiment 4



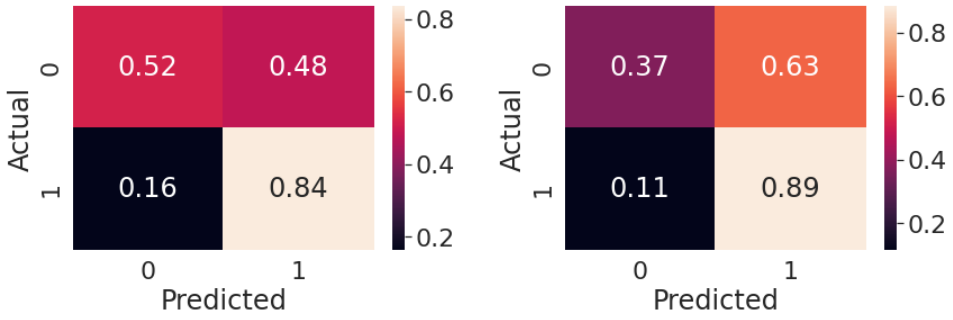
(a) ROC curve **(b)** PR curve

Figure 4.17: ROC and PR curves for depression



(a) ROC curve **(b)** PR curve

Figure 4.18: ROC and PR curves for CVDs



(a) Depression **(b)** CVD

Figure 4.19: Confusion matrices for experiment 4

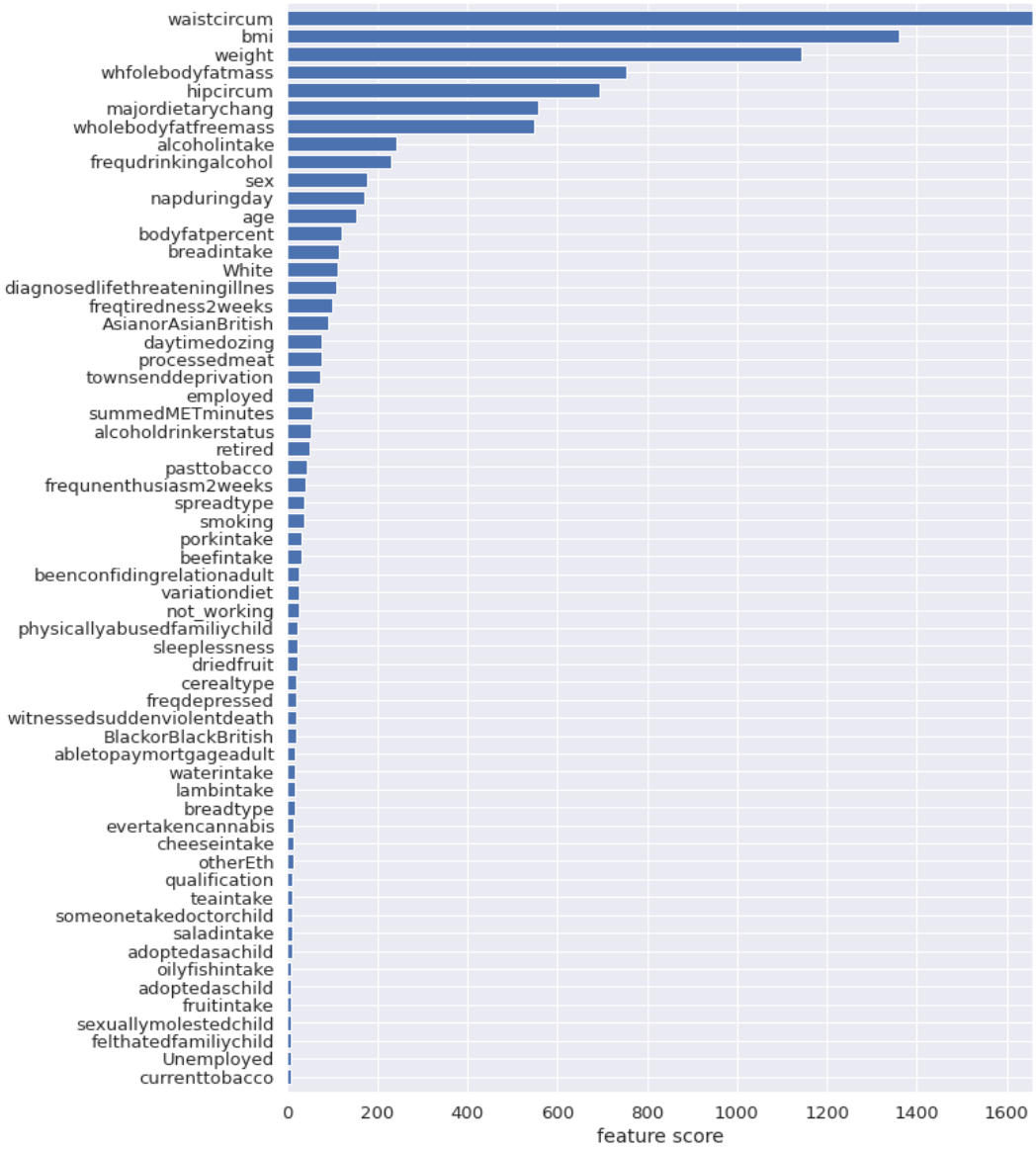


Figure 4.20: Feature selection results for experiment 4

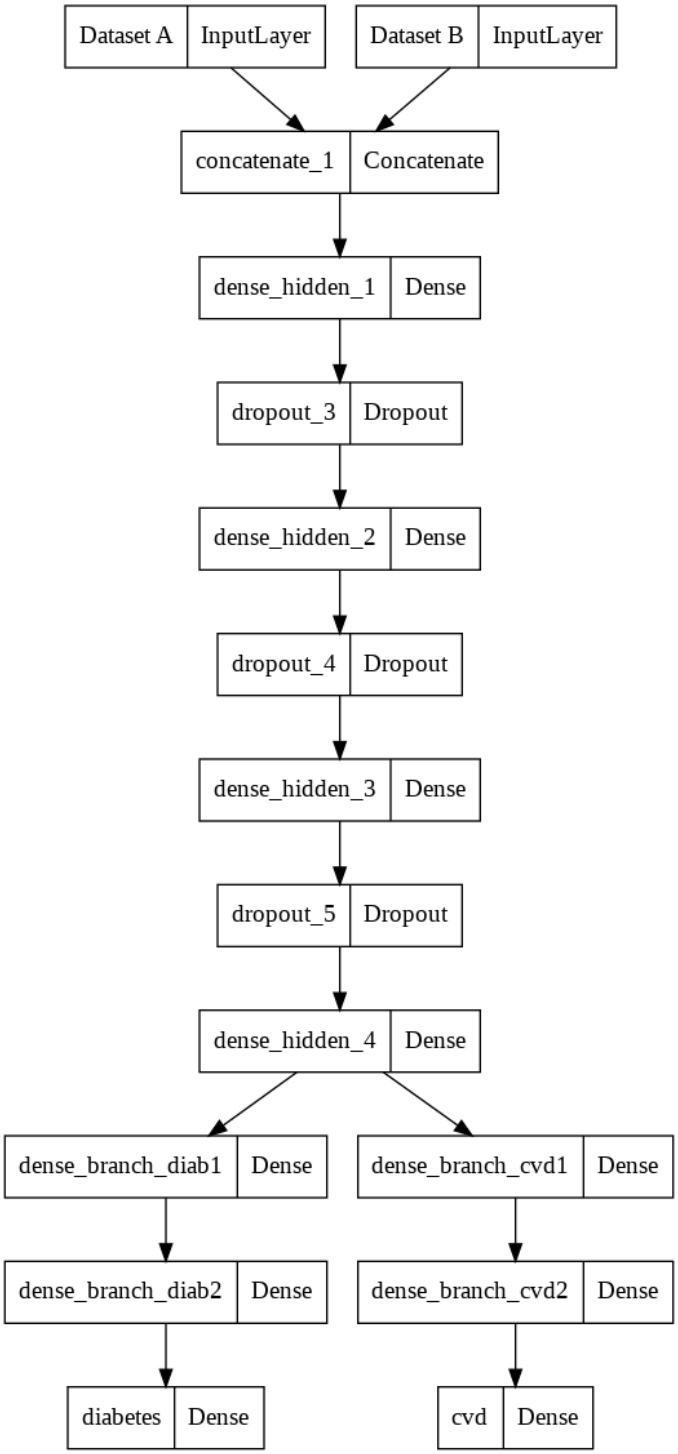


Figure 4.21: MTL model for experiment 4

Chapter 5

Discussion and conclusions

5.1 Interpretation of results

In this work, several metrics were obtained to measure performance of the model. The one that is of prime importance is f-score. This metric indicates how well the model is able to classify test dataset into 1 or 0 class. For each set of input features, the model predicts the occurrence of each considered disease. Each branch of the considered experiment independently predicts probability of disease occurrence. There was originally a huge imbalance in the dataset particularly in case of diabetes, number of class 1 were significantly lower than those of class 0. To address this problem, new synthetically created data points were added to the training dataset using smote technique. However, model was still biased towards learning class 0. In this scenario, performance of the model could not be relied solely on accuracy. Through threshold moving on each considered disease individually, a threshold was chosen so to have best f-score. As indicated by best performance metrics in the previous chapter, the model separates well the test subjects between both classes with an average f-score of 70%. Performance of the model varies among different experiments and the diseases within each experiment.

An important point to note from the results in the previous chapter is that prediction accuracy for each disease is different in each experiment. For example, accuracy for predicting CVDs is 71.4% in experiment 4 versus 60.4% in experiment 1. This variation in results for each disease could be because of the fact that, in order for MTL to perform at its best, considered task must have some degree of correlation. High correlation among tasks would result in improved learning efficiency and prediction accuracy for the task-specific models, when compared to training the models separately. High performance of MTL model in case of experiment 4 could be because of high comormidity among depression and CVDs

in the considered dataset. Model performance in experiment 3 is at its least, which could be because of least correlation among diabetes and CVDs in the considered dataset.

5.2 Limitations

Predicting the occurrence of depressive and cardiometabolic disorders only from exposome features could be an easy and quick tool for physicians to foresee if a person is prone to developing these diseases. However, from the perspective of the model, learning from considered features is not so easy. Despite trying multiple model configurations and hyperparameters, performance of the model could not be improved. Another limitation in performance of the model has been due to a high imbalance in the dataset. Although this problem was addressed through techniques like oversampling and threshold moving but results did not change significantly. More subjects that have the considered diseases (class 1 in the target variable) would improve the model performance.

Due to these limitations of the model, results can not be completely relied upon. However, these could be taken as a preliminary and quick way of predicting the considered diseases. Later, more reliable medical examinations should be carried out to make the diagnosis.

5.3 Future work

The project serves as a preliminary study to diagnose diseases and their comorbidity. At the moment only UK Biobank dataset has been used. It could be useful to consider other datasets to compare the results of this work and also to improve performance of the model. As mentioned earlier, considered features are not sufficient for the model to well separate the data points. Inclusion of more exposome features could result in better model performance. Considering the available results, neural networks based MTL model is a promising tool to study correlated diseases. For an improved performance, a larger dataset with a balanced data could be used in future so that results are more reliable.

5.4 Conclusion

Motivation of this thesis was to leverage from the multitask learning technique to predict diabetes, depression and CVDs using only exposome features. Many techniques were used to make the dataset clean and impute missing values. Initially

no feature selection and oversampling was used which resulted in low performance of the model. However, results improved after selecting 60 most important features and also by exploiting SMOTE-NC to oversample the data. Considered diseases included depression, diabetes and CVDs. In order to study the behaviour of the model, several experiments were conducted with different combinations among the considered diseases. Prediction accuracy of the diseases was different in each experiment due to difference in the commonalities of features. By introducing the improvements suggested in previous section, performance could be improved and the model could be used as a screening tool to diagnose diabetes, depression and CVDs.

Appendix A

Technicalities

A.1 Jupyter notebooks

Two Jupyter files developed during the project have are appended here. These files include all the experiments that have been presented in this work.

1. **Data preprocessing.ipynb** contains code for loading the dataset from csv file to its final stages. Main steps performed include:
   (a) Load the dataset from csv file and select only specified columns
   (b) Filter the columns that contain ICD9 and ICD10 codes for diabetes, depression and CVDs
   (c) Create 3 target features/columns named diabetes, depression and CVD and populate them by consolidating data from other columns
   (d) Remove the columns that are no more required
   (e) Replace -3 (Prefer not to answer) and -1 (do not know) with nan values
   (f) Split the dataset into Dataset A and Dataset B, because for 23 columns we have only limited data
   (g) Impute missing values
   (h) Encode categorical features as a one-hot numeric arrays
   (i) Save the dataframes as csv files
2. **model.ipynb** contains the code for MTL model development and obtaining model performance evaluation metrics. Main steps performed include:
   (a) Loading dataset A and B

(b) Feature selection using SKlearn's SelectKBest function

(c) Obtain and plot score of selected best features

(d) Develop and train the model

(e) Plot the ROC and PR curves of predictions

Data preprocessing

```
import json
import numpy as np
from sklearn.preprocessing import OneHotEncoder
with open('var.json') as f:
    var_temp = json.load(f)

# Adding to the JSON the ICD codes fields
icd10 = []
icd9 = []

for i in range(75):
    var_temp["41202-0."+ str(i)] = "ICD10_"+str(i)
    icd10.append("ICD10_"+str(i))

for i in range(28):
    var_temp["41203-0."+ str(i)] = "ICD9_"+str(i)
    icd9.append("ICD9_"+str(i))

import pandas as pd

df = pd.read_csv('ukb46359.csv', usecols = var_temp.keys()) # just the
columns to select, the ones in the json
df = df.rename(columns = var_temp) # rename the columns with the field
of the json
df.shape
```

```
/opt/anaconda3/envs/bigdatalab_cpu_202101/lib/python3.7/site-
packages/IPython/core/interactiveshell.py:3156: DtypeWarning: Columns
(2843,2844,2845,2846,2847,2848,2849,2850,2851,2852,2853,2854,2855,2856
,2857,2858,2859,2860,2861,2862,2863,2864,2865,2866,2867,2868,2869,2870
,2871,2872,2873,2874,2875,2876,2877,2878,2879,2880,2881,2882,2883,2884
,2885,2886,2887,2888,2889,2890,2891,2892,2893,2894,2895,2896,2897,2898
,2899,2900,2901,2902,2903,2904,2905,2906,2907,2908,2909,2910,2911,2912
,2913,2914,2915,2916,2917,2918,2919,2924) have mixed types.Specify
dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

```
(502481, 328)
```

```
#Add the new dataset containing age and merge it with the previous one
df_add = pd.read_csv('ukb49112.csv', usecols = ['eid', '34-0.0','189-
0.0','1767-0.0','1767-1.0','1767-2.0','1767-3.0'])
df_add['adoptedasachild'] = df_add[['1767-0.0','1767-1.0','1767-
2.0','1767-3.0']].replace({-3:None,-1:None}).max(axis=1)
df_add = df_add.drop(['1767-0.0','1767-1.0','1767-2.0','1767-3.0'],
axis = 1)
```

```
df_add = df_add.rename(columns = {'eid' : 'id', '34-0.0': 'age', '189-
0.0':'townsenddeprivation'})

df = df.merge(df_add, how='left', on='id')
df.shape
```

```
(502481, 331)
```

```
df_ = df.copy()

#sum the MET values to obtain a single column
MET_cols = ["METminutesperweekforwalking",
"METminutesperweekformoderateactivity",
"METminutesperweekforvigorousactivity"]
df_['summedMETminutes']= df_[MET_cols].sum(axis=1)

#replace -7: None of the above with 7 and -3: Prefer not to answer
with nan in qualification columns
#7 as qualification will represent another category
for a in range(6):
    df_["qualifications."+str(a)].replace({-7: 7, -3: 'nan'},
inplace=True)
#Add values in qualification columns to obtain a single column

qual_cols =
["qualifications.0","qualifications.1","qualifications.2","qualificati
ons.3","qualifications.4","qualifications.5"]
df_['qualification']= df_[qual_cols].sum(axis=1)

df_.shape
```

```
(502481, 333)
```

```
# IC_code_cvds --> look the ones that I'm interested

# ICD10 Diseases Of The Circulatory System: I00-I99
ic10_code_cvd_ = ['I0' + str(i) for i in range(10)]
ic10_code_cvd_.extend(['I' + str(i) for i in range(10,100)])
#remove codes related to hypertension
ic10_code_cvd = [e for e in ic10_code_cvd_ if e not in ('I10',
'I11','I12', 'I13', 'I14', 'I15', 'I16')]
# ICD9 Diseases Of The Circulatory System: 390-459
ic9_code_cvd_ = list(range(390,460))
#remove codes related to hypertension
ic9_code_cvd = [e for e in ic9_code_cvd_ if e not in (401, 402, 403,
404, 405, 406)]

df_['cvd_disease_from_icd'] = 0

value = ''*len(df_)
```

```
value_cvd = []

#check for icd10 diagnosis
for col in icd10:
    value += df_[col].map(str)

value = list(value)

for l in range(len(value)):
    #check for CVD
    if any(code in value[l] for code in ic10_code_cvd):

        value_cvd.append(1)
    else:
        value_cvd.append(0)

df_['cvd_disease_from_icd'] = value_cvd

#check for icd9 diagnosis

#find a rows which contain any of icd9 diagnosis code for cvd
df1 = df_.loc[np.where(df_.loc[:,icd9].isin(ic9_code_cvd)==True)[0]]
#add 1 to 'cvd_disease_from_icd' to the corresponding row
df_.loc[df1.index, 'cvd_disease_from_icd'] = 1

# IC_code_depression

#ICD10 code list for Mood [affective] disorders: F30-F39
ic10_code_depr = ['F' + str(i) for i in range(30,40)]
#ic10_code_depr.extend(['F' + str(i) for i in range(10,100)])

#ICD9 code list for Mental Disorders: 290-319
ic9_code_depr = list(range(290,320))

df_['depr_disease_from_icd'] = 0

value = ''*len(df_)
value_depr = []

for col in icd10:
    value += df_[col].map(str)

value = list(value)

for l in range(len(value)):
    #check for Depression/mental disorders
    if any(code in value[l] for code in ic10_code_depr):

        value_depr.append(1)
```

```
    else:
        value_depr.append(0)

df_['depr_disease_from_icd'] = value_depr

#check for icd9 diagnosis

#find a rows which contain any of icd9 diagnosis code for depression
df1 = df_.loc[np.where(df_.loc[:,icd9].isin(ic9_code_depr)==True)[0]]
#add 1 to 'cvd_disease_from_icd' to the corresponding row
df_.loc[df1.index, 'depr_disease_from_icd'] = 1

# IC_code_diabetes

#ICD10 code list for Diabetes mellitus: E08-E13
ic10_code_diab = ['E08', 'E09', 'E10', 'E11', 'E13']

#ICD9 code list for Diabetes mellitus: 250
ic9_code_diab = [250]

df_['diab_disease_from_icd'] = 0

value = ''*len(df_)
value_diab = []

for col in icd10:
    value += df_[col].map(str)

value = list(value)

for l in range(len(value)):
    #check for Depression/mental disorders
    if any(code in value[l] for code in ic10_code_diab):

        value_diab.append(1)
    else:
        value_diab.append(0)

df_['diab_disease_from_icd'] = value_diab

#check for icd9 diagnosis

#find a rows which contain any of icd9 diagnosis code for diabetes
df1 = df_.loc[np.where(df_.loc[:,icd9].isin(ic9_code_diab)==True)[0]]
#add 1 to 'cvd_disease_from_icd' to the corresponding row
df_.loc[df1.index, 'diab_disease_from_icd'] = 1

#include other columns for diagnosis of depression, which include:
#    "2090-0.0":"seendoctordepranxietytime0", values: [ 1.  0. -1. -3.
```

```
nan]
#    "2090-1.0":"seendoctordepranxietytime1"
#    "2090-2.0":"seendoctordepranxietytime2"
#    "2090-3.0":"seendoctordepranxietytime3"
#    "2100-0.0":"seenpsychidepranxietytime0", values: [ 1.  0. -3. -1.
nan]
#    "2100-1.0":"seenpsychidepranxietytime1",
#    "2100-2.0":"seenpsychidepranxietytime2",
#    "2100-3.0":"seenpsychidepranxietytime3",
#    "20126-0.0":"Bipolarandmajordepressionstatus", values: [ 3. nan
0.  4.  5.  2.  1.]
#    "20448-0.0":"Professionalinformedaboutdepression", values: [   1.
nan    0. -818. -121.]
#    "20546-0.1": "medicinedepression1", values: [   1.,   nan,    3.,
4., -818.]
#    "20546-0.2": "medicinedepression2",
#    "20546-0.3": "medicinedepression3",
#add a new column for depression
df_['depression'] = 0
# add 1 to the column depression if the below conditions meet
df_.loc[(df_['seendoctordepranxietytime0'] == 1) |
(df_['seendoctordepranxietytime1'] == 1) |
(df_['seendoctordepranxietytime2'] == 1) |
(df_['seendoctordepranxietytime3'] == 1) |
(df_['seenpsychidepranxietytime0'] == 1) |
(df_['seenpsychidepranxietytime1'] == 1) |
(df_['seenpsychidepranxietytime2'] == 1) |
(df_['seenpsychidepranxietytime3'] == 1) |
(df_['Professionalinformedaboutdepression'] == 1) |
(df_['Bipolarandmajordepressionstatus'].isin([1,2,3,4,5])),'depression
'] = 1
#additional columns related to diagnosis of depression
df_.loc[(df_['medicinedepression1'].isin([1,3,4])) |
(df_['medicinedepression2'].isin([1,3,4])) |
(df_['medicinedepression3'].isin([1,3,4])) , 'depression'] = 1

#As a final step, add 1 to column 'depression' where
'depr_disease_from_icd' = 1
df_.loc[(df_['depr_disease_from_icd'] == 1), 'depression'] = 1

#include other columns for diagnosis of diabetes, which include:
#    "2443-1.0":"diabetesdiagnosistime0", values; [ 0.  1. -1. -3.
nan]
#    "2443-2.0":"diabetesdiagnosistime1", values; [ 0.  1. -1. -3.
nan]
#    "2443-3.0":"diabetesdiagnosistime2", values; [ 0.  1. -1. -3.
nan]
#    "2443-4.0":"diabetesdiagnosistime3", values; [ 0.  1. -1. -3.
nan]
#    "2986-0.0":"Startedinsulinwithinoneyeardiagnosisofdiabetes",
```

```
values: [nan  0.  1. -1. -3.]

#add a new column for diabetes
df_['diabetes'] = 0

df_.loc[(df_["diabetesdiagnosistime0"] == 1) |
(df_["diabetesdiagnosistime1"] == 1) | (df_["diabetesdiagnosistime2"]
== 1) | (df_["diabetesdiagnosistime3"] == 1) |
(df_['Startedinsulinwithinoneyeardiagnosisofdiabetes'] == 1),
'diabetes'] = 1

#As a final step, add 1 to column 'diabetes' where
'diab_disease_from_icd' = 1
df_.loc[(df_['diab_disease_from_icd'] == 1), 'diabetes'] = 1

#include other columns for diagnosis of cardiovascular disease, which
include:
#    "6150-0.0": "cvd0.time0", values: [-7.  1.  4.  2. -3.  3. nan]
#    "6150-0.0": "cvd0.time0",
#    "6150-0.1": "cvd1.time0",
#    "6150-0.2": "cvd2.time0",
#    "6150-0.3": "cvd3.time0",
#    "6150-1.0": "cvd0.time1",
#    "6150-1.1": "cvd1.time1",
#    "6150-1.2": "cvd2.time1",
#    "6150-1.3": "cvd3.time1",
#    "6150-2.0": "cvd0.time2",
#    "6150-2.1": "cvd1.time2",
#    "6150-2.2": "cvd2.time2",
#    "6150-2.3": "cvd3.time2",

#add a new column for diabetes
df_['cvd'] = 0

cvd_cols = ["cvd0.time0", "cvd1.time0","cvd2.time0",
"cvd3.time0","cvd0.time1","cvd1.time1", "cvd2.time1",
            "cvd3.time1", "cvd0.time2", "cvd1.time2", "cvd2.time2",
"cvd3.time2"]

df1 = df_.loc[np.where(df_.loc[:,cvd_cols].isin([1,2,3,4])==True)[0]]
df_.loc[df1.index, 'cvd'] = 1

#Add 1 to column 'cvd' where 'cvd_disease_from_icd' = 1
df_.loc[(df_['cvd_disease_from_icd'] == 1), 'cvd'] = 1

#########################################################################
#########################
# ADD A COLUMN FOT CVDS FROM QUESTIONNAIRE
#########################################################################
#########################
```

```
from_other = ["disease" + str(i) + ".time0" for i in range(30)] #
columns to check
#these are codes of the disease that were self reported by the
subjects
cvd_from_other =
['1066','1067','1068','1074','1075','1076','1077','1078','1079','1080'
,'1081','1082','1083','1086','1087','1088','1093','1094']

#find a rows which contain any of cvd_from_other code in any of
from_other columns
df1 =
df_.loc[np.where(df_.loc[:,from_other].isin(cvd_from_other)==True)[0]]
#add 1 to 'cvd' in the corresponding row
df_.loc[df1.index, 'cvd'] = 1

# codes for diabetes

diab_from_other = ['1220','1222','1223']

#find a rows which contain any of diab_from_other code in any of
from_other columns
df1 =
df_.loc[np.where(df_.loc[:,from_other].isin(diab_from_other)==True)
[0]]
#add 1 to 'cvd' in the corresponding row
df_.loc[df1.index, 'diabetes'] = 1

# codes of depression
depr_from_other = ['1286','1287']

#find a rows which contain any of depr_from_other code in any of
from_other columns
df1 =
df_.loc[np.where(df_.loc[:,from_other].isin(depr_from_other)==True)
[0]]
#add 1 to 'cvd' in the corresponding row
df_.loc[df1.index, 'depression'] = 1

#print(df_['depr_disease_from_icd'].value_counts())
print(df_['depression'].value_counts())
#print(df_['diab_disease_from_icd'].value_counts())
print(df_['diabetes'].value_counts())
#print(df_['cvd_disease_from_icd'].value_counts())
print(df_['cvd'].value_counts())
```

```
0    304958
1    197523
Name: depression, dtype: int64
0    473480
1     29001
```

```
Name: diabetes, dtype: int64
0    286121
1    216360
Name: cvd, dtype: int64
```

```
df_c = df_.copy()
```

```
# Remove rows with cvd missing data and diabetes diagnosis
check = ["disease" + str(i) + ".time0" for i in range(30)]
check.extend(["cvd0.time0","cvd1.time0","cvd2.time0","cvd3.time0"])
df_c = df_c.dropna(how = 'all', subset = check)
```

```
check = ["disease" + str(i) + ".time0" for i in range(30)]
check.extend(["diabetesdiagnosistime0"])
df_c = df_c.dropna(how = 'all', subset = check)
```

```
len(df_c)
```

```
501948
```

```
print(df_c['depression'].value_counts())
print(df_c['diabetes'].value_counts())
print(df_c['cvd'].value_counts())
```

```
0    304449
1    197499
Name: depression, dtype: int64
0    472955
1     28993
Name: diabetes, dtype: int64
0    285718
1    216230
Name: cvd, dtype: int64
```

```
#Now that columns for 3 target diseases are ready,drop the unnecessary
columns from the dataframe
cols = ["seendoctordepranxietytime0",
        "seendoctordepranxietytime1",
        "seendoctordepranxietytime2",
        "seendoctordepranxietytime3",
        "seenpsychidepranxietytime0",
        "seenpsychidepranxietytime1",
        "seenpsychidepranxietytime2",
        "seenpsychidepranxietytime3",
        "Bipolarandmajordepressionstatus",
        "Professionalinformedaboutdepression",
        "medicinedepression1",
        "medicinedepression2",
        "medicinedepression3",
        "diabetesdiagnosistime0",
        "diabetesdiagnosistime1",
```

```
        "diabetesdiagnosistime2",
        "diabetesdiagnosistime3",
        "Startedinsulinwithinoneyeardiagnosisofdiabetes",
        "cvd_disease_from_icd",
        "depr_disease_from_icd",
        "diab_disease_from_icd"]

#getting column names related to questionnaire
# Code for this feature in UKBB startes with 20002, so these column
names can be filtered
col_others = [key[1] for key in var_temp.items() if '20002' in key[0]]

col_to_drop = icd9 + icd10 + cvd_cols + MET_cols + qual_cols + cols +
col_others
df_c = df_c.drop(col_to_drop, axis = 1)
df_c.shape
```

```
(501948, 95)
```

```
for col in df_c.columns:
    print(col)
```

```
id
sex
waistcircum
hipcircum
agecompleted
sleepduration
napduringday
sleeplessness
daytimedozing
currenttobacco
pasttobacco
cookedvegetable
saladintake
fruitintake
driedfruit
oilyfishintake
nonoilyfish
processedmeat
poultryintake
beefintake
lambintake
porkintake
cheeseintake
milkused
spreadtype
breadintake
breadtype
cerealintake
cerealtype
```

salttofood
teaintake
coffetype
waterintake
majordietarychang
variationdiet
alcoholintake
breastfedbaby
adoptedaschild
maternalsmoking
freqdepressed
frequnenthusiasm2weeks
freqtensenness2weeks
freqtiredness2weeks
nonbutterspread
diastolicbloodpress
systolicbloodpress
everdepressed
perioddepress
numberdepressper
everdesinteresedweek
currentemployment
smoking
alcoholdrinkerstatus
neuroticismscore
amountalcoholdrunk
everhadknownpersonalcoh
everbeeninjuredrinkalcohol
frequdrinkingalcohol
evertakencannabis
felthatedfamiliychild
physicallyabusedfamiliychild
feltlovedchild
sexuallymolestedchild
someonetakedoctorchild
avoidedactsstresspasthmonth
repeatedisturbingpastmonth
feltupsetstressfulpastmonth
belittlementbypartneradult
beenconfidingrelationadult
physicalviolenceadult
sexualinterferenceadult
abletopaymortgageadult
beenseriousaccidentlifethrea
diagnosedlifethreateningillnes
victimphysicallyviolentcrime
witnessedsuddenviolentdeath
victimsexualassault
ethnic
bmi

```
weight
bodyfatpercent
whfolebodyfatmass
wholebodyfatfreemass
Creactive
hardphysicalactivity
moderatephysicalactivity
softphysicalactivity
age
townsenddeprivation
adoptedasachild
summedMETminutes
qualification
depression
diabetes
cvd

#Replace -3: Prefer not to answer and -1: do not know, with nan values
#Replace -7 which represents 'None of the above' in qualification
column. 8 can represent a new category
#Replace -10 which represents 'less than 1' for integer columns with 1
#Replace -818 which represents 'Prefer not to answer' with None
#Replace -2 which represents 'never went to school' with 1 (just a
small number)
for col in df_c.columns:
    df_c[col].replace({-3: np.nan, -1: np.nan, -2:1, -7:8, -10:1, -
818: np.nan}, inplace=True)
    #print(str(col), df_c[col].isna().sum())

df_c.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 501948 entries, 0 to 502480
Data columns (total 95 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   id                              501948 non-null  int64
 1   sex                             501948 non-null  float64
 2   waistcircum                     500161 non-null  float64
 3   hipcircum                       500102 non-null  float64
 4   agecompleted                    330607 non-null  float64
 5   sleepduration                   498235 non-null  float64
 6   napduringday                    500559 non-null  float64
 7   sleeplessness                   500947 non-null  float64
 8   daytimedozing                   498718 non-null  float64
 9   currenttobacco                  501131 non-null  float64
 10  pasttobacco                     460333 non-null  float64
 11  cookedvegetable                 494861 non-null  float64
 12  saladintake                     494606 non-null  float64
 13  fruitintake                     499212 non-null  float64
 14  driedfruit                      495495 non-null  float64
```

```
15  oilyfishintake                  498375 non-null  float64
16  nonoilyfish                     498732 non-null  float64
17  processedmeat                   500224 non-null  float64
18  poultryintake                   500382 non-null  float64
19  beefintake                      499131 non-null  float64
20  lambintake                      497968 non-null  float64
21  porkintake                      498161 non-null  float64
22  cheeseintake                    487999 non-null  float64
23  milkused                        500930 non-null  float64
24  spreadtype                      496653 non-null  float64
25  breadintake                     491670 non-null  float64
26  breadtype                       479795 non-null  float64
27  cerealintake                    499372 non-null  float64
28  cerealtype                      412001 non-null  float64
29  salttofood                      501330 non-null  float64
30  teaintake                       500264 non-null  float64
31  coffetype                       384378 non-null  float64
32  waterintake                     497675 non-null  float64
33  majordietarychang               500092 non-null  float64
34  variationdiet                   495269 non-null  float64
35  alcoholintake                   500955 non-null  float64
36  breastfedbaby                   383684 non-null  float64
37  adoptedaschild                  499976 non-null  float64
38  maternalsmoking                 432831 non-null  float64
39  freqdepressed                   478295 non-null  float64
40  frequnenthusiasm2weeks          482661 non-null  float64
41  freqtensenness2weeks            480274 non-null  float64
42  freqtiredness2weeks             485217 non-null  float64
43  nonbutterspread                 258424 non-null  float64
44  diastolicbloodpress             467900 non-null  float64
45  systolicbloodpress              467887 non-null  float64
46  everdepressed                   168163 non-null  float64
47  perioddepress                   71766 non-null   float64
48  numberdepressper                64274 non-null   float64
49  everdesinteresedweek            164006 non-null  float64
50  currentemployment               499482 non-null  float64
51  smoking                         499504 non-null  float64
52  alcoholdrinkerstatus            500803 non-null  float64
53  neuroticismscore                401545 non-null  float64
54  amountalcoholdrunk              143099 non-null  float64
55  everhadknownpersonalcoh         157055 non-null  float64
56  everbeeninjuredrinkalcohol      157201 non-null  float64
57  frequdrinkingalcohol            157099 non-null  float64
58  evertakencannabis               157065 non-null  float64
59  felthatedfamiliychild           156840 non-null  float64
60  physicallyabusedfamiliychild    156936 non-null  float64
61  feltlovedchild                  156666 non-null  float64
62  sexuallymolestedchild           155442 non-null  float64
63  someonetakedoctorchild          156220 non-null  float64
64  avoidedactsstresspasthmonth     157016 non-null  float64
```

```
 65  repeatedisturbingpastmonth      157058 non-null  float64
 66  feltupsetstressfulpastmonth     157049 non-null  float64
 67  belittlementbypartneradult      156809 non-null  float64
 68  beenconfidingrelationadult      153169 non-null  float64
 69  physicalviolenceadult           156803 non-null  float64
 70  sexualinterferenceadult         156779 non-null  float64
 71  abletopaymortgageadult          154846 non-null  float64
 72  beenseriousaccidentlifethrea    157085 non-null  float64
 73  diagnosedlifethreateningillnes  156656 non-null  float64
 74  victimphysicallyviolentcrime    156976 non-null  float64
 75  witnessedsuddenviolentdeath     157013 non-null  float64
 76  victimsexualassault             155297 non-null  float64
 77  ethnic                          499681 non-null  float64
 78  bmi                             499217 non-null  float64
 79  weight                          492199 non-null  float64
 80  bodyfatpercent                  491919 non-null  float64
 81  whfolebodyfatmass               491355 non-null  float64
 82  wholebodyfatfreemass            492151 non-null  float64
 83  Creactive                       468394 non-null  float64
 84  hardphysicalactivity            70700 non-null   float64
 85  moderatephysicalactivity        70700 non-null   float64
 86  softphysicalactivity            70700 non-null   float64
 87  age                             501887 non-null  float64
 88  townsenddeprivation             501266 non-null  float64
 89  adoptedasachild                 499974 non-null  float64
 90  summedMETminutes                501948 non-null  float64
 91  qualification                   501948 non-null  float64
 92  depression                      501948 non-null  int64
 93  diabetes                        501948 non-null  int64
 94  cvd                             501948 non-null  int64
dtypes: float64(91), int64(4)
memory usage: 367.6 MB
```

```
els1 =  ['amountalcoholdrunk',
         'everhadknownpersonalcoh',
         'everbeeninjuredrinkalcohol',
         'frequdrinkingalcohol',
         'evertakencannabis',
         'felthatedfamiliychild',
         'physicallyabusedfamiliychild',
         'feltlovedchild',
         'sexuallymolestedchild',
         'someonetakedoctorchild',
         'avoidedactsstresspasthmonth',
         'repeatedisturbingpastmonth',
         'feltupsetstressfulpastmonth',
         'belittlementbypartneradult',
         'beenconfidingrelationadult',
         'physicalviolenceadult',
         'sexualinterferenceadult',
```

```
        'abletopaymortgageadult',
        'beenseriousaccidentlifethrea',
        'diagnosedlifethreateningillnes',
        'victimphysicallyviolentcrime',
        'witnessedsuddenviolentdeath',
        'victimsexualassault']


print("len of df_c: ", len(df_c.columns))
print("len of els1: ", len(els1))
cols = df_c.columns.tolist()

for col in els1:
        cols.remove(col)
#these two dataframes will be our 2 separate datasets
df_ca = df_c[els1]
df_cb = df_c[cols]

print("df_ca shape: ", df_ca.shape)
print("df_cb shape: ", df_cb.shape)
```

```
len of df_c:  95
len of els1:  23
df_ca shape:  (501948, 23)
df_cb shape:  (501948, 72)
```

```
#copy target disease columns and merge them with df_small dataset
df_small = df_ca.dropna(how = 'all')
print("df_small shape: ", df_small.shape)
targets = df_c.loc[df_small.index]
targets = targets[['diabetes', 'depression', 'cvd']].copy()
df_small = pd.concat([df_small, targets],
axis=1).reindex(df_small.index)
print("df_small shape: ", df_small.shape)
```

```
df_small shape:  (157287, 23)
df_small shape:  (157287, 26)
```

```
#Remove any row or column in the df_cb that is completely nan
df_big = df_cb.dropna(how = 'all')
print("df_big shape: ", df_big.shape)
```

```
df_big shape:  (501948, 72)
```

```
df_temp = df_big.copy()

els2 = ['hardphysicalactivity',
        'moderatephysicalactivity',
        'softphysicalactivity']

df_temp = df_temp.loc[(df_temp['cvd']==0) & (df_temp['diabetes']==0) &
```

```
(df_temp['depression']==0)]
df_temp = df_temp.loc[(df_temp['hardphysicalactivity']==0) &
(df_temp['moderatephysicalactivity']==0) &
(df_temp['softphysicalactivity']==0)]
df_big = df_big.drop(df_temp.index)
df_big.shape
```

```
(501238, 72)
```

```
print(df_big['depression'].value_counts())
print(df_big['diabetes'].value_counts())
print(df_big['cvd'].value_counts())
```

```
0    303739
1    197499
Name: depression, dtype: int64
0    472245
1     28993
Name: diabetes, dtype: int64
0    285008
1    216230
Name: cvd, dtype: int64
```

```
#dictionary specifying the data type i.c. categorical or int or cont
for each column
feat_cat = {'sex': 'cat',
 'waistcircum': 'cont',
 'hipcircum': 'cont',
 'sleepduration': 'int',
 'napduringday': 'cat',
 'sleeplessness': 'cat',
 'daytimedozing': 'cat',
 'currenttobacco': 'cat',
 'cookedvegetable': 'int',
 'saladintake': 'int',
 'fruitintake': 'int',
 'driedfruit': 'int',
 'oilyfishintake': 'cat',
 'nonoilyfish': 'cat',
 'processedmeat': 'cat',
 'poultryintake': 'cat',
 'beefintake': 'cat',
 'lambintake': 'cat',
 'porkintake': 'cat',
 'cheeseintake': 'cat',
 'milkused': 'cat',
 'spreadtype': 'cat',
 'breadintake': 'int',
 'breadtype': 'cat',
 'cerealintake': 'int',
 'salttofood': 'cat',
```

```
'teaintake': 'int',
'waterintake': 'int',
'majordietarychang': 'cat',
'variationdiet': 'cat',
'alcoholintake': 'cat',
'adoptedaschild': 'cat',
'freqdepressed': 'cat',
'frequnenthusiasm2weeks': 'cat',
'freqtensenness2weeks': 'cat',
'freqtiredness2weeks': 'cat',
'currentemployment': 'cat',
'smoking': 'cat',
'alcoholdrinkerstatus': 'cat',
'ethnic': 'cat',
'bmi': 'cont',
'weight': 'cont',
'bodyfatpercent': 'cont',
'whfolebodyfatmass': 'cont',
'wholebodyfatfreemass': 'cont',
'summedMETminutes': 'cont',
'qualification': 'cat',
'agecompleted': 'int',
'pasttobacco': 'cat',
'cerealtype': 'cat',
'coffetype': 'cat',
'breastfedbaby': 'cat',
'maternalsmoking': 'cat',
'nonbutterspread': 'cat',
'depression': 'cat',
'diabetes': 'cat',
'cvd': 'cat',
'someonetakedoctorchild': 'cat',
'sexuallymolestedchild': 'cat',
'feltlovedchild': 'cat',
'physicallyabusedfamiliychild': 'cat',
'felthatedfamiliychild': 'cat',
'amountalcoholdrunk': 'cat',
'everhadknownpersonalcoh': 'cat',
'everbeeninjuredrinkalcohol': 'cat',
'frequdrinkingalcohol': 'cat',
'evertakencannabis': 'cat',
'avoidedactsstresspasthmonth': 'cat',
'repeatedisturbingpastmonth': 'cat',
'feltupsetstressfulpastmonth': 'cat',
'belittlementbypartneradult': 'cat',
'beenconfidingrelationadult': 'cat',
'physicalviolenceadult': 'cat',
'sexualinterferenceadult': 'cat',
'abletopaymortgageadult': 'cat',
'beenseriousaccidentlifethrea': 'cat',
```

```
 'diagnosedlifethreateningillnes': 'cat',
 'victimphysicallyviolentcrime': 'cat',
 'witnessedsuddenviolentdeath': 'cat',
 'victimsexualassault': 'cat',
 'hardphysicalactivity': 'cat',
 'moderatephysicalactivity': 'cat',
 'softphysicalactivity': 'cat',
 'age': 'int',
 'townsenddeprivation': 'cont',
 'adoptedasachild': 'cat'
          }

len(feat_cat)
```

```
86
```

```
more_cols_drop = [ 'id','diastolicbloodpress',
 'systolicbloodpress',
 'everdepressed',
 'perioddepress',
 'numberdepressper',
 'everdesinteresedweek',
 'neuroticismscore',
 'Creactive']

df_big = df_big.drop(more_cols_drop, axis = 1)
df_big.shape
```

```
(501238, 63)
```

```
# subtract each value in column 'age' from 2012 to calculate age
df_big['age'] = 2012 - df_big['age']

df_big.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 501238 entries, 0 to 502480
Data columns (total 63 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   sex                       501238 non-null  float64
 1   waistcircum               499452 non-null  float64
 2   hipcircum                 499394 non-null  float64
 3   agecompleted              330142 non-null  float64
 4   sleepduration             497534 non-null  float64
 5   napduringday              499852 non-null  float64
 6   sleeplessness             500241 non-null  float64
 7   daytimedozing             498018 non-null  float64
 8   currenttobacco            500423 non-null  float64
 9   pasttobacco               459677 non-null  float64
 10  cookedvegetable           494170 non-null  float64
```

```
11  saladintake               493912 non-null  float64
12  fruitintake               498510 non-null  float64
13  driedfruit                494800 non-null  float64
14  oilyfishintake            497675 non-null  float64
15  nonoilyfish               498031 non-null  float64
16  processedmeat             499521 non-null  float64
17  poultryintake             499679 non-null  float64
18  beefintake                498432 non-null  float64
19  lambintake                497271 non-null  float64
20  porkintake                497461 non-null  float64
21  cheeseintake              487314 non-null  float64
22  milkused                  500224 non-null  float64
23  spreadtype                495948 non-null  float64
24  breadintake               490970 non-null  float64
25  breadtype                 479117 non-null  float64
26  cerealintake              498666 non-null  float64
27  cerealtype                411445 non-null  float64
28  salttofood                500621 non-null  float64
29  teaintake                 499557 non-null  float64
30  coffetype                 383860 non-null  float64
31  waterintake               496969 non-null  float64
32  majordietarychang         499388 non-null  float64
33  variationdiet             494568 non-null  float64
34  alcoholintake             500250 non-null  float64
35  breastfedbaby             383144 non-null  float64
36  adoptedaschild            499269 non-null  float64
37  maternalsmoking           432212 non-null  float64
38  freqdepressed             477614 non-null  float64
39  frequnenthusiasm2weeks    481989 non-null  float64
40  freqtensenness2weeks      479603 non-null  float64
41  freqtiredness2weeks       484527 non-null  float64
42  nonbutterspread           258102 non-null  float64
43  currentemployment         498779 non-null  float64
44  smoking                   498798 non-null  float64
45  alcoholdrinkerstatus      500098 non-null  float64
46  ethnic                    498973 non-null  float64
47  bmi                       498513 non-null  float64
48  weight                    491498 non-null  float64
49  bodyfatpercent            491219 non-null  float64
50  whfolebodyfatmass         490657 non-null  float64
51  wholebodyfatfreemass      491450 non-null  float64
52  hardphysicalactivity      69990 non-null   float64
53  moderatephysicalactivity  69990 non-null   float64
54  softphysicalactivity      69990 non-null   float64
55  age                       501177 non-null  float64
56  townsenddeprivation       500559 non-null  float64
57  adoptedasachild           499267 non-null  float64
58  summedMETminutes          501238 non-null  float64
59  qualification             501238 non-null  float64
60  depression                501238 non-null  int64
```

```
 61  diabetes                  501238 non-null  int64
 62  cvd                       501238 non-null  int64
dtypes: float64(60), int64(3)
memory usage: 244.7 MB
```

Managing missing values

Fill using meadian, mode and mean

```
# use mode for categorical, mean for continuous and mode for integer
type values

#--------for df_big-----------------
# fill_mbig --> Dataset A
fill_mbig = df_big.copy()
for col in fill_mbig.columns:
    if feat_cat[col] == 'cat' or feat_cat[col] == 'int' or
feat_cat[col] == 'num':
        fill_mbig[col] = fill_mbig[col].fillna(fill_mbig[col].mode()
[0])
    elif feat_cat[col] == 'cont':
        fill_mbig[col] = fill_mbig[col].fillna(fill_mbig[col].mean())
    else:
        print(col + " not in dict")

fill_mbig.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 501238 entries, 0 to 502480
Data columns (total 63 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   sex                       501238 non-null  float64
 1   waistcircum               501238 non-null  float64
 2   hipcircum                 501238 non-null  float64
 3   agecompleted              501238 non-null  float64
 4   sleepduration             501238 non-null  float64
 5   napduringday              501238 non-null  float64
 6   sleeplessness             501238 non-null  float64
 7   daytimedozing             501238 non-null  float64
 8   currenttobacco            501238 non-null  float64
 9   pasttobacco               501238 non-null  float64
 10  cookedvegetable           501238 non-null  float64
 11  saladintake               501238 non-null  float64
 12  fruitintake               501238 non-null  float64
 13  driedfruit                501238 non-null  float64
 14  oilyfishintake            501238 non-null  float64
 15  nonoilyfish               501238 non-null  float64
 16  processedmeat             501238 non-null  float64
 17  poultryintake             501238 non-null  float64
 18  beefintake                501238 non-null  float64
```

```
 19  lambintake                501238 non-null  float64
 20  porkintake                501238 non-null  float64
 21  cheeseintake              501238 non-null  float64
 22  milkused                  501238 non-null  float64
 23  spreadtype                501238 non-null  float64
 24  breadintake               501238 non-null  float64
 25  breadtype                 501238 non-null  float64
 26  cerealintake              501238 non-null  float64
 27  cerealtype                501238 non-null  float64
 28  salttofood                501238 non-null  float64
 29  teaintake                 501238 non-null  float64
 30  coffetype                 501238 non-null  float64
 31  waterintake               501238 non-null  float64
 32  majordietarychang         501238 non-null  float64
 33  variationdiet             501238 non-null  float64
 34  alcoholintake             501238 non-null  float64
 35  breastfedbaby             501238 non-null  float64
 36  adoptedaschild            501238 non-null  float64
 37  maternalsmoking           501238 non-null  float64
 38  freqdepressed             501238 non-null  float64
 39  frequnenthusiasm2weeks    501238 non-null  float64
 40  freqtensenness2weeks      501238 non-null  float64
 41  freqtiredness2weeks       501238 non-null  float64
 42  nonbutterspread           501238 non-null  float64
 43  currentemployment         501238 non-null  float64
 44  smoking                   501238 non-null  float64
 45  alcoholdrinkerstatus      501238 non-null  float64
 46  ethnic                    501238 non-null  float64
 47  bmi                       501238 non-null  float64
 48  weight                    501238 non-null  float64
 49  bodyfatpercent            501238 non-null  float64
 50  whfolebodyfatmass         501238 non-null  float64
 51  wholebodyfatfreemass      501238 non-null  float64
 52  hardphysicalactivity      501238 non-null  float64
 53  moderatephysicalactivity  501238 non-null  float64
 54  softphysicalactivity      501238 non-null  float64
 55  age                       501238 non-null  float64
 56  townsenddeprivation       501238 non-null  float64
 57  adoptedasachild           501238 non-null  float64
 58  summedMETminutes          501238 non-null  float64
 59  qualification             501238 non-null  float64
 60  depression                501238 non-null  int64
 61  diabetes                  501238 non-null  int64
 62  cvd                       501238 non-null  int64
dtypes: float64(60), int64(3)
memory usage: 244.7 MB
```

```
# use mode for categorical, mean for continuous and mode for integer
type values
```

```
#--------for df_small-----------------
# fill_msmall --> Dataset B
fill_msmall = df_small.copy()
for col in fill_msmall.columns:
    if feat_cat[col] == 'cat' or feat_cat[col] == 'int' or
feat_cat[col] == 'num':
        fill_msmall[col] =
fill_msmall[col].fillna(fill_msmall[col].mode()[0])
    elif feat_cat[col] == 'cont':
        fill_msmall[col] =
fill_msmall[col].fillna(fill_msmall[col].mean())
    else:
        print(col + " not in dict")

fill_msmall.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 157287 entries, 0 to 502479
Data columns (total 26 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   amountalcoholdrunk              157287 non-null  float64
 1   everhadknownpersonalcoh         157287 non-null  float64
 2   everbeeninjuredrinkalcohol      157287 non-null  float64
 3   frequdrinkingalcohol            157287 non-null  float64
 4   evertakencannabis               157287 non-null  float64
 5   felthatedfamiliychild           157287 non-null  float64
 6   physicallyabusedfamiliychild    157287 non-null  float64
 7   feltlovedchild                  157287 non-null  float64
 8   sexuallymolestedchild           157287 non-null  float64
 9   someonetakedoctorchild          157287 non-null  float64
 10  avoidedactsstresspasthmonth     157287 non-null  float64
 11  repeatedisturbingpastmonth      157287 non-null  float64
 12  feltupsetstressfulpastmonth     157287 non-null  float64
 13  belittlementbypartneradult      157287 non-null  float64
 14  beenconfidingrelationadult      157287 non-null  float64
 15  physicalviolenceadult           157287 non-null  float64
 16  sexualinterferenceadult         157287 non-null  float64
 17  abletopaymortgageadult          157287 non-null  float64
 18  beenseriousaccidentlifethrea    157287 non-null  float64
 19  diagnosedlifethreateningillnes  157287 non-null  float64
 20  victimphysicallyviolentcrime    157287 non-null  float64
 21  witnessedsuddenviolentdeath     157287 non-null  float64
 22  victimsexualassault             157287 non-null  float64
 23  diabetes                        157287 non-null  int64
 24  depression                      157287 non-null  int64
 25  cvd                             157287 non-null  int64
dtypes: float64(23), int64(3)
memory usage: 32.4 MB
```

```
#Mangaging categorical values
def manage_cat(df_ca):

    #In 'currenttobacco' column, switch 1:Yes, on most or all days and
2:Only occasionally to maintain an order
    df_ca['currenttobacco'].replace({1: 2, 2: 1}, inplace=True)

    # 'currentemployment' has following categories:
    # 1:In paid employment or self-employed
    # 2:Retired
    # 3:Looking after home and/or family
    # 4:Unable to work because of sickness or disability
    # 5:Unemployed
    # 6:Doing unpaid or voluntary work
    # 7:Full or part-time student
    # 8:None of the above

    #Some of them can be merged to simplify the columns as:
    # 0:Unemployed & Full or part-time student
    # 1:Looking after home and/or family & Doing unpaid or voluntary
work & Unable to work because of sickness or disability & other
    # 2:Retired
    # 3:In paid employment or self-employed

    newvalsemp = {1:3, 3:1, 4:1, 5:0, 6:1, 7:0, 8:1}
    print("df_ca['currentemployment'] unique: ",
df_ca['currentemployment'].unique())
    df_ca['currentemployment'] =
df_ca['currentemployment'].replace(newvalsemp)

    # 'ethnic' has many categories but they can be merged together to
obtain fewer categories (using only top level categories in UKBB for
this feature)
    # New categories could be like this:
    # 1 (White): 1001 & 1002 & 1003
    # 2 (Mixed): 2001 & 2002 & 2003 & 2004
    # 3 (Asian or Asian British): 3001 & 3002 & 3003 & 3004 & 5
    # 4 (Black or Black British): 4001 & 4002 & 4003
    # 5 (other): 6


    newvalseth = {1001:1, 1002:1, 1003:1, 2001:2, 2002:2, 2003:2,
2004:2, 3001:3, 3002:3, 3003:3, 3004:3,
                  5:3, 4001:4, 4002:4, 4003:4, 6:5}
    df_ca['ethnic'] = df_ca['ethnic'].replace(newvalseth)

    # 'majordietarychang' has following categories:
    # 0:No
    # 1:Yes, on most or all days
    # 2:Only occasionally
```

```
    # 1 and 2 can be merged for simplification
    df_ca['majordietarychang'].replace({2: 1}, inplace=True)

    # creating instance of one-hot-encoder
    enc = OneHotEncoder(handle_unknown='ignore')
    # passing currentemployment column (label encoded values of
currentemployment)
    enc_emp =
pd.DataFrame(enc.fit_transform(df_ca[['currentemployment']]).toarray()
)

    #update new column names
    employment_status = ['Unemployed', 'not_working', 'retired',
'employed']
    enc_emp   = enc_emp.set_axis(employment_status, axis=1,
inplace=False)
    enc_emp.reset_index(drop=True, inplace=True)# reset the index of
the dataframe
    # passing ethnic column (label encoded values of ethnic)
    enc_eth =
pd.DataFrame(enc.fit_transform(df_ca[['ethnic']]).toarray())
    print(enc_emp.isna().sum())
    print("enc_emp.shape: ", enc_emp.shape)

    #update new column names
    ethnicities = ['White', 'Mixed', 'AsianorAsianBritish',
'BlackorBlackBritish', 'otherEth']
    enc_eth   = enc_eth.set_axis(ethnicities, axis=1, inplace=False)
    enc_eth.reset_index(drop=True, inplace=True)# reset the index of
the dataframe
    print(enc_eth.isna().sum())
    print("enc_eth.shape: ", enc_eth.shape)
    df_ca.reset_index(drop=True, inplace=True)# reset the index of the
dataframe
    print(df_ca.isna().sum())
    print("df_ca.shape: ", df_ca.shape)
    # merge new dataframes ie enc_emp and enc_eth with main df on key
values
    df_ca = df_ca.join(enc_emp)
    df_ca = df_ca.join(enc_eth)

    #Remove the columns 'currentemployment' and 'ethnic' from the main
df
    df_ca = df_ca.drop(['currentemployment', 'ethnic'], axis = 1)

    print(df_ca.isna().sum())
    print("df_ca.shape: ", df_ca.shape)

    return df_ca
```

```
fill_mbig = manage_cat(fill_mbig)
fill_mbig.to_csv('fill_mbig.csv', index=False)
```

```
df_ca['currentemployment'] unique:  [1. 2. 5. 8. 4. 3. 7. 6.]
Unemployed     0
not_working    0
retired        0
employed       0
dtype: int64
enc_emp.shape:  (501238, 4)
White                   0
Mixed                   0
AsianorAsianBritish     0
BlackorBlackBritish     0
otherEth                0
dtype: int64
enc_eth.shape:  (501238, 5)
sex                  0
waistcircum          0
hipcircum            0
agecompleted         0
sleepduration        0
                   ..
summedMETminutes     0
qualification        0
depression           0
diabetes             0
cvd                  0
Length: 63, dtype: int64
df_ca.shape:  (501238, 63)
sex                     0
waistcircum             0
hipcircum               0
agecompleted            0
sleepduration           0
                       ..
White                   0
Mixed                   0
AsianorAsianBritish     0
BlackorBlackBritish     0
otherEth                0
Length: 70, dtype: int64
df_ca.shape:  (501238, 70)
```

```
fill_msmall.to_csv('fill_msmall.csv', index=False)
```

Model.ipynb

```
%matplotlib inline
%load_ext tensorboard

import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import shutil
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.layers import Input, Dense
from keras import metrics
from sklearn.metrics import confusion_matrix
from collections import Counter
from sklearn.metrics import f1_score
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import RepeatedStratifiedKFold
from keras.layers import Dropout
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
from matplotlib import pyplot
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import RepeatedStratifiedKFold
import random
from sklearn.utils import shuffle
from imblearn.under_sampling import NearMiss
from imblearn.over_sampling import SMOTENC
import seaborn as sns
from tensorflow.keras.layers import Concatenate, Dense
from imblearn.over_sampling import RandomOverSampler
from sklearn.feature_selection import SelectKBest, chi2
import seaborn as sn

import warnings
warnings.filterwarnings('ignore')

try:
  shutil.rmtree('logs')
except:
  pass
```

```
#dfS --> Dataset B
dfS =pd.read_csv('/content/drive/MyDrive/Thesis/fill_msmall.csv')
dfS
```

```
        amountalcoholdrunk  everhadknownpersonalcoh  \
0                      2.0                      1.0
1                      1.0                      0.0
2                      2.0                      0.0
3                      2.0                      0.0
4                      2.0                      0.0
...                    ...                      ...
157282                 2.0                      0.0
157283                 1.0                      0.0
157284                 1.0                      0.0
157285                 3.0                      0.0
157286                 1.0                      0.0

        everbeeninjuredrinkalcohol  frequdrinkingalcohol
evertakencannabis  \
0                              0.0                   4.0
3.0
1                              0.0                   4.0
0.0
2                              0.0                   3.0
0.0
3                              0.0                   2.0
0.0
4                              0.0                   2.0
3.0
...                            ...                   ...
...
157282                         0.0                   3.0
0.0
157283                         0.0                   4.0
3.0
157284                         0.0                   1.0
1.0
157285                         0.0                   3.0
0.0
157286                         0.0                   1.0
0.0

        felthatedfamiliychild  physicallyabusedfamiliychild
feltlovedchild  \
0                         2.0                           2.0
2.0
1                         0.0                           0.0
4.0
2                         0.0                           0.0
4.0
```

```
3                          0.0                              0.0
4.0
4                          0.0                              0.0
4.0
...                        ...                              ...
...
157282                     0.0                              0.0
4.0
157283                     1.0                              0.0
3.0
157284                     1.0                              2.0
1.0
157285                     0.0                              0.0
4.0
157286                     0.0                              0.0
3.0

        sexuallymolestedchild  someonetakedoctorchild  ...  \
0                         0.0                     4.0  ...
1                         0.0                     4.0  ...
2                         0.0                     4.0  ...
3                         1.0                     4.0  ...
4                         0.0                     4.0  ...
...                       ...                     ...  ...
157282                    1.0                     4.0  ...
157283                    0.0                     4.0  ...
157284                    3.0                     4.0  ...
157285                    0.0                     4.0  ...
157286                    0.0                     4.0  ...

        sexualinterferenceadult  abletopaymortgageadult  \
0                           0.0                     3.0
1                           0.0                     4.0
2                           0.0                     4.0
3                           0.0                     4.0
4                           0.0                     4.0
...                         ...                     ...
157282                      0.0                     4.0
157283                      0.0                     4.0
157284                      0.0                     3.0
157285                      0.0                     4.0
157286                      0.0                     4.0

        beenseriousaccidentlifethrea
diagnosedlifethreateningillnes  \
0                                0.0                              0.0

1                                0.0                              0.0

2                                1.0                              0.0
```

```
3                                1.0                                0.0

4                                0.0                                0.0

...                              ...                                ...

157282                           0.0                                0.0

157283                           1.0                                0.0

157284                           0.0                                0.0

157285                           0.0                                0.0

157286                           0.0                                0.0


        victimphysicallyviolentcrime  witnessedsuddenviolentdeath  \
0                                0.0                          0.0
1                                0.0                          0.0
2                                1.0                          1.0
3                                1.0                          1.0
4                                0.0                          0.0
...                              ...                          ...
157282                           0.0                          0.0
157283                           1.0                          0.0
157284                           1.0                          0.0
157285                           0.0                          0.0
157286                           0.0                          0.0

        victimsexualassault  diabetes  depression  cvd
0                       0.0         0           1    1
1                       0.0         0           0    1
2                       0.0         0           1    1
3                       1.0         0           1    1
4                       0.0         0           0    0
...                     ...       ...         ...  ...
157282                  0.0         0           0    1
157283                  0.0         0           1    0
157284                  1.0         0           1    0
157285                  0.0         0           0    1
157286                  0.0         0           0    0

[157287 rows x 26 columns]
```

```
#dfB --> Dataset A
dfB = pd.read_csv('/content/drive/MyDrive/Thesis/fill_mbig.csv')
dfB
```

```
        sex  waistcircum  hipcircum  agecompleted  sleepduration  \
0       0.0         74.0      102.0          16.0            7.0
1       1.0        120.0      113.0          16.0            9.0
2       0.0         66.0       88.0          16.0            5.0
3       1.0        110.0      117.0          18.0            7.0
4       1.0         94.0      100.0          16.0            6.0
...     ...          ...        ...           ...            ...
501233  1.0        110.0      113.0          16.0            7.0
501234  0.0        102.0      110.0          16.0            6.0
501235  1.0         99.0       98.0           1.0            7.0
501236  0.0         86.0      111.0          16.0            6.0
501237  1.0         83.0       98.0          16.0           12.0

        napduringday  sleeplessness  daytimedozing  currenttobacco  \
0                2.0            3.0            0.0             0.0
1                2.0            2.0            0.0             0.0
2                1.0            3.0            0.0             0.0
3                2.0            1.0            0.0             2.0
4                2.0            3.0            0.0             0.0
...              ...            ...            ...             ...
501233           2.0            2.0            0.0             1.0
501234           2.0            3.0            0.0             0.0
501235           2.0            2.0            1.0             0.0
501236           1.0            3.0            0.0             0.0
501237           2.0            3.0            0.0             0.0

        pasttobacco  ...  cvd  Unemployed  not_working  retired
employed  \
0               1.0  ...    1         0.0          0.0      0.0
1.0
1               1.0  ...    1         0.0          0.0      0.0
1.0
2               3.0  ...    0         0.0          0.0      1.0
0.0
3               4.0  ...    1         0.0          0.0      1.0
0.0
4               2.0  ...    1         1.0          0.0      0.0
0.0
...             ...  ...  ...         ...          ...      ...
...
501233          1.0  ...    1         0.0          0.0      0.0
1.0
501234          1.0  ...    1         0.0          0.0      1.0
0.0
501235          1.0  ...    1         0.0          1.0      0.0
0.0
501236          4.0  ...    0         0.0          0.0      0.0
1.0
501237          3.0  ...    1         0.0          0.0      0.0
1.0
```

```
        White  Mixed  AsianorAsianBritish  BlackorBlackBritish
otherEth
0         1.0    0.0                  0.0                  0.0
0.0
1         1.0    0.0                  0.0                  0.0
0.0
2         1.0    0.0                  0.0                  0.0
0.0
3         1.0    0.0                  0.0                  0.0
0.0
4         1.0    0.0                  0.0                  0.0
0.0
...       ...    ...                  ...                  ...       .
..
501233    1.0    0.0                  0.0                  0.0
0.0
501234    1.0    0.0                  0.0                  0.0
0.0
501235    1.0    0.0                  0.0                  0.0
0.0
501236    1.0    0.0                  0.0                  0.0
0.0
501237    1.0    0.0                  0.0                  0.0
0.0

[501238 rows x 70 columns]
```

```
# dfm --> subset of dataset containing all features (to be used for
feature selection)
dfm = pd.read_csv('/content/drive/MyDrive/Thesis/fill_m.csv')
dfm
```

```
        sex  waistcircum  hipcircum  agecompleted  sleepduration  \
0       0.0         74.0      102.0          16.0            7.0
1       1.0         83.0       92.0          16.0            7.0
2       1.0         93.0      107.0          16.0            8.0
3       0.0         67.0       91.0          18.0            8.0
4       1.0         94.0      105.0          16.0            7.0
...     ...          ...        ...           ...            ...
157067  0.0         83.0      103.0          16.0            7.0
157068  1.0         80.0      101.0          16.0            7.0
157069  0.0         70.0       92.0          16.0            9.0
157070  1.0        110.0      113.0          16.0            7.0
157071  0.0         86.0      111.0          16.0            6.0

        napduringday  sleeplessness  daytimedozing  currenttobacco  \
0                2.0            3.0            0.0             0.0
1                2.0            2.0            1.0             0.0
2                1.0            2.0            0.0             0.0
```

```
3                2.0           1.0           0.0            0.0
4                1.0           1.0           0.0            0.0
...              ...           ...           ...            ...
157067           1.0           3.0           0.0            0.0
157068           1.0           1.0           0.0            0.0
157069           1.0           2.0           0.0            0.0
157070           2.0           2.0           0.0            1.0
157071           1.0           3.0           0.0            0.0

        pasttobacco  ...  cvd  Unemployed  not_working  retired
employed  \
0               1.0  ...    1         0.0          0.0      0.0
1.0
1               1.0  ...    1         0.0          0.0      1.0
0.0
2               3.0  ...    1         0.0          0.0      0.0
1.0
3               4.0  ...    1         0.0          0.0      0.0
1.0
4               2.0  ...    0         0.0          0.0      0.0
1.0
...             ...  ...  ...         ...          ...      ...
...
157067          4.0  ...    1         0.0          0.0      0.0
1.0
157068          2.0  ...    0         0.0          0.0      0.0
1.0
157069          4.0  ...    0         0.0          0.0      0.0
1.0
157070          1.0  ...    1         0.0          0.0      0.0
1.0
157071          4.0  ...    0         0.0          0.0      0.0
1.0

        White  Mixed  AsianorAsianBritish  BlackorBlackBritish
otherEth
0         1.0    0.0                  0.0                  0.0
0.0
1         1.0    0.0                  0.0                  0.0
0.0
2         1.0    0.0                  0.0                  0.0
0.0
3         1.0    0.0                  0.0                  0.0
0.0
4         1.0    0.0                  0.0                  0.0
0.0
...       ...    ...                  ...                  ...       .
..
157067    1.0    0.0                  0.0                  0.0
0.0
```

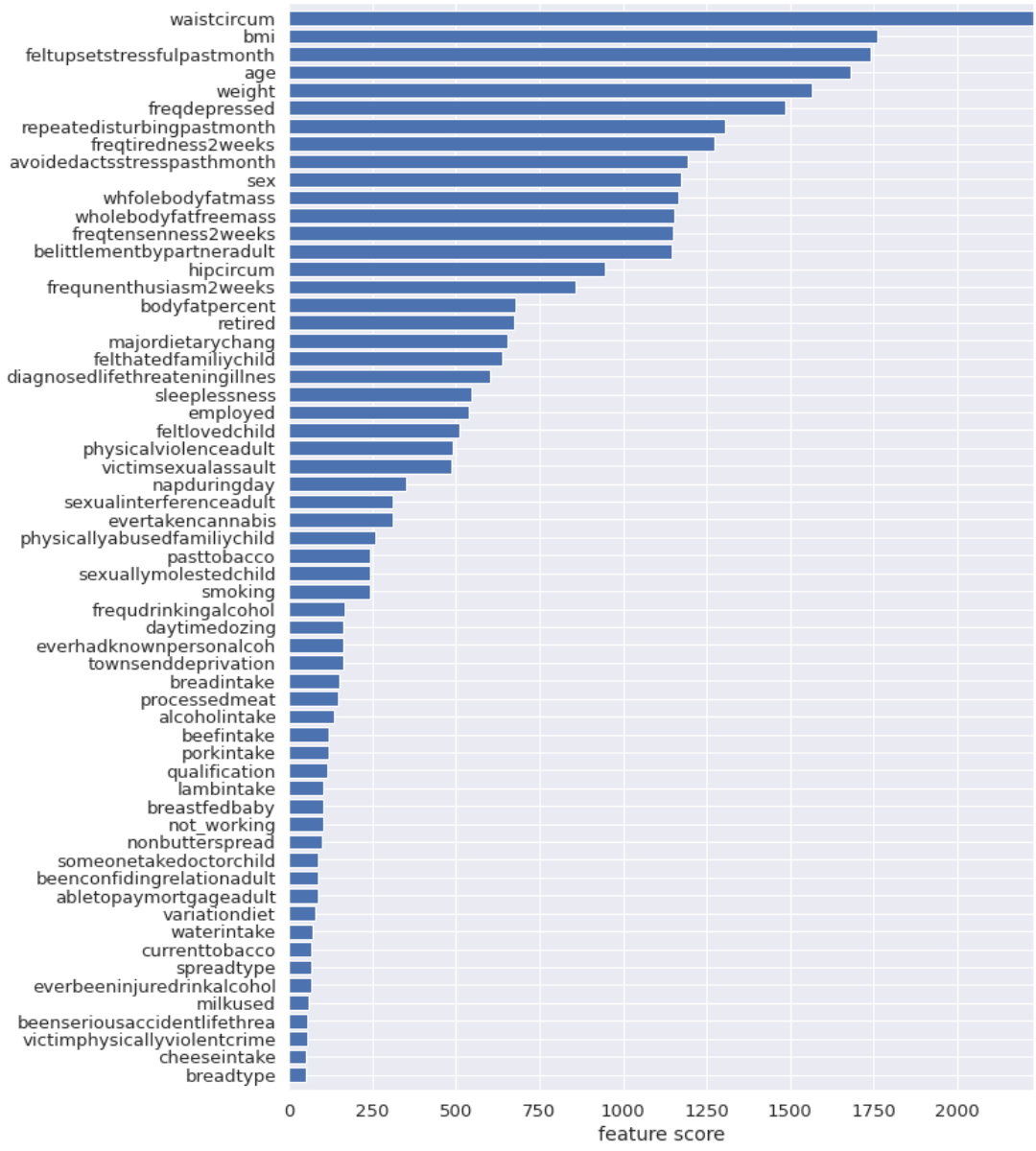
```
157068    1.0    0.0                 0.0                 0.0
0.0
157069    1.0    0.0                 0.0                 0.0
0.0
157070    1.0    0.0                 0.0                 0.0
0.0
157071    1.0    0.0                 0.0                 0.0
0.0

[157072 rows x 93 columns]
```

Experiment 1: Diabetes, depression and CVDs

Select best features

```
df = dfm.copy()
df['all'] =  df['diabetes'].astype(str) + df['depression'].astype(str)
+df['cvd'].astype(str)
#reset the indexes of the newly created dataframe
df.reset_index(drop=True, inplace=True)

cols = df.columns.tolist()
cols.remove('diabetes')
cols.remove('depression')
cols.remove('cvd')
cols.remove('all')
df_X = df[cols]
df_Y = df[['all']]

# define feature selection
fs = SelectKBest(k=60).fit(df_X,df_Y)

X_selected = fs.transform(df_X)

# Get columns to keep and create new dataframe with those only
b_cols = fs.get_support(indices=True)
df_new = df_X.iloc[:,b_cols]
bestcols = df_new.columns.to_list()
scores = fs.scores_

li = df_X.columns.to_list()
scoresl = scores.tolist()
bestscore = []
for each in bestcols:
  bestscore.append(scoresl[li.index(each)])

res = {}
for each in bestcols:
  res[each] = bestscore[bestcols.index(each)]
```

```
ress = {k: v for k, v in sorted(res.items(), key=lambda item:
item[1])}

plt.figure(figsize=(10,15))
plt.margins(x=0, y=0.008)
plt.xlabel("feature score")
plt.barh(list(ress.keys()), list(ress.values()))
sn.set(font_scale=1.2)
plt.show()
```




```
#dictionary specifying the data type i.c. categorical or int or cont
for each column
feat_cat = {'sex': 'cat',
```

```
'waistcircum': 'cont',
'hipcircum': 'cont',
'sleepduration': 'int',
'napduringday': 'cat',
'sleeplessness': 'cat',
'daytimedozing': 'cat',
'currenttobacco': 'cat',
'cookedvegetable': 'int',
'saladintake': 'int',
'fruitintake': 'int',
'driedfruit': 'int',
'oilyfishintake': 'cat',
'nonoilyfish': 'cat',
'processedmeat': 'cat',
'poultryintake': 'cat',
'beefintake': 'cat',
'lambintake': 'cat',
'porkintake': 'cat',
'cheeseintake': 'cat',
'milkused': 'cat',
'spreadtype': 'cat',
'breadintake': 'int',
'breadtype': 'cat',
'cerealintake': 'int',
'salttofood': 'cat',
'teaintake': 'int',
'waterintake': 'int',
'majordietarychang': 'cat',
'variationdiet': 'cat',
'alcoholintake': 'cat',
'adoptedaschild': 'cat',
'freqdepressed': 'cat',
'frequnenthusiasm2weeks': 'cat',
'freqtensenness2weeks': 'cat',
'freqtiredness2weeks': 'cat',
'currentemployment': 'cat',
'smoking': 'cat',
'alcoholdrinkerstatus': 'cat',
'ethnic': 'cat',
'bmi': 'cont',
'weight': 'cont',
'bodyfatpercent': 'cont',
'whfolebodyfatmass': 'cont',
'wholebodyfatfreemass': 'cont',
'summedMETminutes': 'cont',
'qualification': 'cat',
'agecompleted': 'int',
'pasttobacco': 'cat',
'cerealtype': 'cat',
'coffetype': 'cat',
```

```
 'breastfedbaby': 'cat',
 'maternalsmoking': 'cat',
 'nonbutterspread': 'cat',
 'depression': 'cat',
 'diabetes': 'cat',
 'cvd': 'cat',
 'someonetakedoctorchild': 'cat',
 'sexuallymolestedchild': 'cat',
 'feltlovedchild': 'cat',
 'physicallyabusedfamiliychild': 'cat',
 'felthatedfamiliychild': 'cat',
 'amountalcoholdrunk': 'cat',
 'everhadknownpersonalcoh': 'cat',
 'everbeeninjuredrinkalcohol': 'cat',
 'frequdrinkingalcohol': 'cat',
 'evertakencannabis': 'cat',
 'avoidedactsstresspasthmonth': 'cat',
 'repeatedisturbingpastmonth': 'cat',
 'feltupsetstressfulpastmonth': 'cat',
 'belittlementbypartneradult': 'cat',
 'beenconfidingrelationadult': 'cat',
 'physicalviolenceadult': 'cat',
 'sexualinterferenceadult': 'cat',
 'abletopaymortgageadult': 'cat',
 'beenseriousaccidentlifethrea': 'cat',
 'diagnosedlifethreateningillnes': 'cat',
 'victimphysicallyviolentcrime': 'cat',
 'witnessedsuddenviolentdeath': 'cat',
 'victimsexualassault': 'cat',
 'hardphysicalactivity': 'cat',
 'moderatephysicalactivity': 'cat',
 'softphysicalactivity': 'cat',
 'Unemployed': 'cat',
 'not_working': 'cat',
 'retired': 'cat',
 'employed': 'cat',
 'White': 'cat',
 'Mixed': 'cat',
 'AsianorAsianBritish': 'cat',
 'BlackorBlackBritish': 'cat',
 'otherEth': 'cat',
 'age': 'int',
 'townsenddeprivation': 'cont',
 'adoptedasachild': 'cat'
          }

cols = dfB.columns.tolist()
cols.remove('diabetes')
cols.remove('depression')
cols.remove('cvd')
```

```
#get indexes of categorical features
categorical_featB = []
for each in cols:
    if feat_cat[each] == 'cat':
        categorical_featB.append(cols.index(each))

print("-----Preparing initial datasets - Train and holdout-----")
cols = dfB.columns.tolist()
cols.remove('diabetes')
cols.remove('depression')
cols.remove('cvd')

colsB = []
for each in cols:
  if each in bestcols:
    colsB.append(each)


df_XB = dfB[colsB]
df_YB = dfB[['diabetes', 'depression', 'cvd']]
cols = dfS.columns.tolist()
cols.remove('diabetes')
cols.remove('depression')
cols.remove('cvd')

colsS = []
for each in cols:
  if each in bestcols:
    colsS.append(each)

df_XS = dfS[colsS]
df_YS = dfS[['diabetes', 'depression', 'cvd']]

#get indexes of categorical features
categorical_featB = []
for each in colsB:
    if feat_cat[each] == 'cat':
        categorical_featB.append(colsB.index(each))

nB_cols = len(colsB)
nS_cols = len(colsS)

#prepare train and test dataset manually from the bigger dataset
test_size  = int(df_XB.shape[0] * 0.02)
df1 = df_YB.copy()
df1['out'] = df1['diabetes'].astype(str) +
df1['depression'].astype(str) +df1['cvd'].astype(str)
dis_cols  = df1['out'].unique().tolist()
```

```
rowsfortestB = []
for each in dis_cols:
  tempY = df1[df1.out == each]
  rows = np.random.choice(tempY.index.values,
int(test_size/len(dis_cols)))
  rowsfortestB.append(rows)

rowsfortestB = [j for i in rowsfortestB for j in i]
X_testB = df_XB.loc[rowsfortestB]
y_testB = df_YB.loc[rowsfortestB]
X_trainB = df_XB.loc[~df_XB.index.isin(rowsfortestB)]
y_trainB= df_YB.loc[~df_YB.index.isin(rowsfortestB)]

print("\nDistribution of Dataset B")
print("shape of X_trainB: ", X_trainB.shape)
print("shape of y_trainB: ", y_trainB.shape)
print("shape of X_testB: ", X_testB.shape)
print("shape of y_testB: ", y_testB.shape)

#prepare train and test dataset manually from the small dataset
df1S = df_YS.copy()
df1S['out'] = df1S['diabetes'].astype(str) +
df1S['depression'].astype(str) +df1S['cvd'].astype(str)

rowsfortestS = []
for each in dis_cols:
  tempY = df1S[df1S.out == each]
  rows = np.random.choice(tempY.index.values,
int(test_size/len(dis_cols)))
  rowsfortestS.append(rows)

rowsfortestS = [j for i in rowsfortestS for j in i]
X_testS = df_XS.loc[rowsfortestS]
y_testS = df_YS.loc[rowsfortestS]
X_trainS = df_XS.loc[~df_XS.index.isin(rowsfortestS)]
y_trainS = df_YS.loc[~df_YS.index.isin(rowsfortestS)]

print("\nDistribution of Dataset S")
print("shape of X_trainS: ", X_trainS.shape)
print("shape of y_trainS: ", y_trainS.shape)
print("shape of X_testS: ", X_testS.shape)
print("shape of y_testS: ", y_testS.shape)



#for larger dataset (A)
# perform a scaler transform on train and test dataset
trans = StandardScaler()
trans.fit(X_trainB)
```

```
data1 = trans.transform(X_trainB)
X_trainB = pd.DataFrame(data1)
#Apply same parameters of standardization on test as on train
data2 = trans.transform(X_testB)
X_testB = pd.DataFrame(data2)

#for smaller dataset (B)
# perform a scaler transform on train and test dataset
trans1 = StandardScaler()
trans1.fit(X_trainS)

data1 = trans1.transform(X_trainS)
X_trainS = pd.DataFrame(data1)
#Apply same parameters of standardization on test as on train
data2 = trans1.transform(X_testS)
X_testS = pd.DataFrame(data2)


# Model configuration
batch_size = 500
no_epochs = 5
verbosity = 1
num_folds = 5

# Define per-fold score containers
acc_per_fold = []
loss_per_fold = []


# Define the K-fold Cross Validator
kfold = StratifiedKFold(n_splits=num_folds, shuffle=True)
rskf = RepeatedStratifiedKFold(n_splits=num_folds, n_repeats=2,
random_state=36851234)

# K-fold Cross Validation model evaluation
fold_no = 1

inputs = X_trainB.copy()
targets = y_trainB.copy()

inputsize = 0

# Define the layers of the model
input_A = Input(shape = (nB_cols,), name = 'Dataset A')
input_B = Input(shape = (nS_cols,), name = 'Dataset B')
out = Concatenate()([input_A, input_B])
dense_hidden_1 = Dense(60, activation = 'relu', name =
'dense_hidden_1')(out)
```

```
dense_hidden_1_d = Dropout(0.5)(dense_hidden_1)
dense_hidden_2 = Dense(55, activation = 'relu', name =
'dense_hidden_2')(dense_hidden_1_d)
dense_hidden_2_d = Dropout(0.25)(dense_hidden_2)
dense_hidden_3 = Dense(50, activation = 'relu', name =
'dense_hidden_3')(dense_hidden_2_d)
dense_hidden_3_d = Dropout(0.1)(dense_hidden_3)
dense_hidden_4 = Dense(45, activation = 'relu', name =
'dense_hidden_4')(dense_hidden_3_d)


dense_branch_diab1 = Dense(40, activation = 'relu', name =
'dense_branch_diab1')(dense_hidden_4)
#dense_hidden_diab1_d = Dropout(0.25)(dense_branch_diab1)
dense_branch_diab2 = Dense(35, activation = 'relu', name =
'dense_branch_diab2')(dense_branch_diab1)
dense_branch_diab3 = Dense(30, activation = 'relu', name =
'dense_branch_diab3')(dense_branch_diab2)
dense_branch_diab4 = Dense(20, activation = 'relu', name =
'dense_branch_diab4')(dense_branch_diab2)
diabetes = Dense(1, activation = 'sigmoid', name = 'diabetes')
(dense_branch_diab4)


dense_branch_depr1 = Dense(40, activation = 'relu', name =
'dense_branch_depr1')(dense_hidden_4)
#dense_hidden_depr1_d = Dropout(0.1)(dense_branch_depr1)
dense_branch_depr2 = Dense(35, activation = 'relu', name =
'dense_branch_depr2')(dense_branch_depr1)
dense_branch_depr3 = Dense(30, activation = 'relu', name =
'dense_branch_depr3')(dense_branch_depr2)
dense_branch_depr4 = Dense(20, activation = 'relu', name =
'dense_branch_depr4')(dense_branch_depr3)
depression = Dense(1, activation = 'sigmoid', name ='depression')
(dense_branch_depr4)

dense_branch_cvd1 = Dense(40, activation = 'relu', name =
'dense_branch_cvd1')(dense_hidden_4)
#dense_hidden_cvd1_d = Dropout(0.25)(dense_branch_cvd1)
dense_branch_cvd2 = Dense(35, activation = 'relu', name =
'dense_branch_cvd2')(dense_branch_cvd1)
dense_branch_cvd3 = Dense(30, activation = 'relu', name =
'dense_branch_cvd3')(dense_branch_cvd2)
dense_branch_cvd4 = Dense(20, activation = 'relu', name =
'dense_branch_cvd4')(dense_branch_cvd3)
cvd = Dense(1, activation = 'sigmoid', name ='cvd')(dense_branch_cvd4)

for train, test in kfold.split(inputs, targets['diabetes']):
```

```
  trainX = inputs.iloc[train]
  trainy = targets.iloc[train]
  testXB = inputs.iloc[test]
  testyB = targets.iloc[test]
  #print("shape of trainX: ", trainX.shape)
  #print("trainy orig unique: \n", trainy.value_counts())

  # split the small dataset into train and validation
  X_train_s, X_tests, y_train_s, y_tests = train_test_split(X_trainS,
y_trainS, test_size=0.20,  stratify=y_trainS['diabetes'])

  #merge the target columns for undersampling
  print("\nDropping 000 rows from the Dataset A")
  df1 = trainy.copy()
  df1['out'] = df1['diabetes'].astype(str) +
df1['depression'].astype(str) +df1['cvd'].astype(str)
  orig_count = df1['out'].value_counts()
  orig_count = orig_count.to_dict()
  #print("orig_count before: ", orig_count)
  print("['out'] count before SMOTE: \n",df1['out'].value_counts())

  #selective undersampling for specific classes
  # reduce samples for 000 to match it with second highest class
  orig_count['000'] = 0
  all_values = orig_count.values()
  orig_count['000'] = int(max(all_values))
  # define the undersampling method
  undersample = NearMiss(version=1, n_neighbors=3, sampling_strategy =
orig_count)
  trainy = df1['out']
  df_US_X, df_US_y = undersample.fit_resample(trainX, trainy)


  print("\nOversampling the dataset B using SMOTE-NC")
  oversample = SMOTENC(categorical_features=categorical_featB)
  df_OS_X, df_OS_y = oversample.fit_resample(df_US_X, df_US_y)


  #split the 'out column to 3 target diseases
  a = pd.DataFrame(df_OS_y)
  a.columns = ['out']
  print("\nClass distribution in training - Dataset B:")
  print(a['out'].value_counts())
  #print("['out'] count after SMOTE: \n",a['out'].value_counts())
  a['diabetes'] = a['out'].str[:1]
  a['depression'] = a['out'].str[1:2]
  a['cvd'] = a['out'].str[2:3]
  df_OS_y = a.drop('out', axis=1)

  #convert datatype of target columns to int
```

```
  df_OS_y['diabetes'] = df_OS_y['diabetes'].astype(str).astype(int)
  df_OS_y['depression'] =
df_OS_y['depression'].astype(str).astype(int)
  df_OS_y['cvd'] = df_OS_y['cvd'].astype(str).astype(int)

  trainXB = df_OS_X.copy()
  trainyB = df_OS_y.copy()



  #create input_S dataset for training
  print("\nPreparing training data for Dataset S")
  m = trainXB.shape[0]
  yb = trainyB.copy()
  yb['all'] = yb['diabetes'].astype(str) +
yb['depression'].astype(str) +yb['cvd'].astype(str)
  #print("y counts for train: \n", ys['all'].value_counts())
  yb_count = yb['all'].value_counts().to_dict()
  #print("yb_count: ", yb_count)
  S_y = y_train_s.copy()
  S_y['all'] = S_y['diabetes'].astype(str) +
S_y['depression'].astype(str) +S_y['cvd'].astype(str)
  S_y_count = S_y['all'].value_counts().to_dict()
  #print("B_y['all'] counts: \n", B_y_count)

  colmov = S_y[['diabetes', 'depression', 'cvd']]
  #print("shape of X_train_s: ", X_train_s.shape)
  #print("shape of S_y: ", S_y.shape)
  X_train_s.reset_index(drop=True, inplace=True)
  colmov.reset_index(drop=True, inplace=True)
  df_XSn = pd.concat([X_train_s, colmov], axis=1, join="inner")
  #print("shape of df_XSn: ", df_XSn.shape)
  #print("shape of B_y: ", B_y.shape)

  S_y_countNew = S_y_count.copy()
  for each in S_y_count:
    if S_y_count[each] > yb_count[each]:
      S_y_countNew[each] = yb_count[each]

  #some classes in S_y have more samples than in ys, drop the extra
samples
  undersample = NearMiss(version=1, n_neighbors=3, sampling_strategy =
S_y_countNew)
  df_S_X, df_S_y = undersample.fit_resample(df_XSn, S_y['all'])

  ros = RandomOverSampler(sampling_strategy = yb_count)

  X_res, y_res = ros.fit_resample(df_S_X, df_S_y)
  X_res = pd.DataFrame(X_res, columns = df_XSn.columns)
```

```
  y_res = pd.DataFrame(y_res)
  y_res.columns = ['all']

  #print("X_res shape: ", X_res.shape)
  #print("y_res shape: ", y_res.shape)
  y_res['newind'] = 0
  for each in yb_count:
    indexes = yb.loc[yb['all']==each].index.tolist()
    y_res.loc[y_res['all'] == each, 'newind'] = indexes
  X_res = pd.concat([X_res, y_res['newind']], axis=1, join="inner")
  X_res = X_res.sort_values('newind')
  X_res.reset_index(drop=True, inplace=True)
  X_res = X_res.drop(['diabetes', 'depression', 'cvd','newind'], axis
= 1)
  trainXS = X_res.copy()

  print("Dataset B train shape: ", trainXB.shape)
  print("Dataset S train shape: ", trainXS.shape)

  #create input_S dataset for validation
  print("\nPreparing training data for Dataset S")
  m = testXB.shape[0]
  yb = testyB.copy()
  yb['all'] = yb['diabetes'].astype(str) +
yb['depression'].astype(str) +yb['cvd'].astype(str)
  #print("y counts for train: \n", ys['all'].value_counts())
  yb_count = yb['all'].value_counts().to_dict()
  #print("yb_count: ", yb_count)
  S_y = y_tests.copy()
  S_y['all'] = S_y['diabetes'].astype(str) +
S_y['depression'].astype(str) +S_y['cvd'].astype(str)
  S_y_count = S_y['all'].value_counts().to_dict()
  #print("B_y['all'] counts: \n", B_y_count)

  colmov = S_y[['diabetes', 'depression', 'cvd']]
  #print("shape of X_train_s: ", X_train_s.shape)
  #print("shape of S_y: ", S_y.shape)
  X_tests.reset_index(drop=True, inplace=True)
  colmov.reset_index(drop=True, inplace=True)
  df_XSn = pd.concat([X_tests, colmov], axis=1, join="inner")
  #print("shape of df_XSn: ", df_XSn.shape)
  #print("shape of B_y: ", B_y.shape)

  S_y_countNew = S_y_count.copy()
  for each in S_y_count:
    if S_y_count[each] > yb_count[each]:
      S_y_countNew[each] = yb_count[each]

  #some classes in S_y have more samples than in ys, drop the extra
samples
```

```
  undersample = NearMiss(version=1, n_neighbors=3, sampling_strategy =
S_y_countNew)
  df_S_X, df_S_y = undersample.fit_resample(df_XSn, S_y['all'])

  ros = RandomOverSampler(sampling_strategy = yb_count)

  X_res, y_res = ros.fit_resample(df_S_X, df_S_y)
  X_res = pd.DataFrame(X_res, columns = df_XSn.columns)
  y_res = pd.DataFrame(y_res)
  y_res.columns = ['all']

  #print("X_res shape: ", X_res.shape)
  #print("y_res shape: ", y_res.shape)
  y_res['newind'] = 0
  for each in yb_count:
    indexes = yb.loc[yb['all']==each].index.tolist()
    y_res.loc[y_res['all'] == each, 'newind'] = indexes
  X_res = pd.concat([X_res, y_res['newind']], axis=1, join="inner")
  X_res = X_res.sort_values('newind')
  X_res.reset_index(drop=True, inplace=True)
  X_res = X_res.drop(['diabetes', 'depression', 'cvd','newind'], axis
= 1)
  testXS = X_res.copy()


  print("Dataset B validation shape: ", testXB.shape)
  print("Dataset S validation shape: ", testXS.shape)

  model = tf.keras.models.Model([input_A, input_B], [diabetes,
depression, cvd])

  initial_learning_rate = 0.1
  lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
      initial_learning_rate,
      decay_steps=10000,
      decay_rate=0.96,
      staircase=True)

  optimizeradam = tf.keras.optimizers.Adam(lr=0.0001)

  model.compile(
      loss={
          'diabetes': 'binary_crossentropy',
          'depression': 'binary_crossentropy',
          'cvd': 'binary_crossentropy'
      },
      optimizer = optimizeradam,
      metrics = ['accuracy']
  )
```

```
  # Generate a print

print('------------------------------------------------------------------
---------')
  print(f'Training for fold {fold_no} ...')
  #print("len(df_OS_X): ", len(df_OS_X))
  #print("len(testX): ", len(testX))

  inputsize += trainXB.shape[0]

  # Fit data to model
  history = model.fit([trainXB, trainXS], [trainyB['diabetes'],
trainyB['depression'],trainyB['cvd']],
            validation_data = ([testXB, testXS], [testyB['diabetes'],
testyB['depression'], testyB['cvd']]),
            batch_size=batch_size,
            epochs=no_epochs,
            verbose=0)

  # Generate generalization metrics
  scores = model.evaluate([testXB, testXS], [testyB['diabetes'],
testyB['depression'], testyB['cvd']], verbose=0)

#  print(f'Score for fold {fold_no}: {model.metrics_names[0]} of
{scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
# acc_per_fold.append(scores[1] * 100)
#  loss_per_fold.append(scores[0])

  # Increase fold number
  fold_no = fold_no + 1


#sort both test datasets to align target disease columns
y_testB.reset_index(drop=True, inplace=True)
X_testB.reset_index(drop=True, inplace=True)
y_testB['all'] = y_testB['diabetes'].astype(str) +
y_testB['depression'].astype(str) +y_testB['cvd'].astype(str)

#create a dataframe for target variables of test B
all_testXB = pd.concat([X_testB, y_testB['all']], axis=1,
join="inner")
all_testXB.sort_values(by=['all'])
a = pd.DataFrame(columns=['all'])
a['all'] = all_testXB['all'].copy()
all_testXB = all_testXB.drop(['all'], axis = 1)
print("\nClass distribution in test - Dataset B:")
print(a['all'].value_counts())
#print("['out'] count after SMOTE: \n",a['out'].value_counts())
a['diabetes'] = a['all'].str[:1]
```

```
a['depression'] = a['all'].str[1:2]
a['cvd'] = a['all'].str[2:3]
all_testyB = a.drop('all', axis=1)
#convert datatype of target columns to int
all_testyB['diabetes'] =
all_testyB['diabetes'].astype(str).astype(int)
all_testyB['depression'] =
all_testyB['depression'].astype(str).astype(int)
all_testyB['cvd'] = all_testyB['cvd'].astype(str).astype(int)


y_testS.reset_index(drop=True, inplace=True)
X_testS.reset_index(drop=True, inplace=True)
y_testS['all'] = y_testS['diabetes'].astype(str) +
y_testS['depression'].astype(str) +y_testS['cvd'].astype(str)
#create a dataframe for target variables of test S
all_testXS = pd.concat([X_testS, y_testS['all']], axis=1,
join="inner")
all_testXS.sort_values(by=['all'])
a = pd.DataFrame(columns=['all'])
a['all'] = all_testXS['all'].copy()
all_testXS = all_testXS.drop(['all'], axis = 1)
print("\nClass distribution in test - Dataset S:")
print(a['all'].value_counts())
#print("['out'] count after SMOTE: \n",a['out'].value_counts())
a['diabetes'] = a['all'].str[:1]
a['depression'] = a['all'].str[1:2]
a['cvd'] = a['all'].str[2:3]
all_testyS = a.drop('all', axis=1)
#convert datatype of target columns to int
all_testyS['diabetes'] =
all_testyS['diabetes'].astype(str).astype(int)
all_testyS['depression'] =
all_testyS['depression'].astype(str).astype(int)
all_testyS['cvd'] = all_testyS['cvd'].astype(str).astype(int)



preds = model.predict([all_testXB, all_testXS])
# Metrics for each target disease
target_v = ['diabetes','depression', 'cvd']

pred_threshold = {}

print("Input size: ", inputsize)

for each in target_v:
  print("\n-------------------------------------------------------")
  print(each)
```

```
  print("-----------------------------------------------------")

  #preds = model.predict(X_test)
  y_true = all_testyB[each]
  predictions = preds[target_v.index(each)]

  lr_probs = predictions

  fpr, tpr, thresholds = roc_curve(y_true, lr_probs)
  # calculate the g-mean for each threshold
  gmeans = np.sqrt(tpr * (1-fpr))
  # locate the index of the largest g-mean
  ix = np.argmax(gmeans)
  print('Best Threshold=%f, G-Mean=%.3f' % (thresholds[ix],
gmeans[ix]))
  # plot the roc curve for the model
  pyplot.plot([0,1], [0,1], linestyle='--', label='No Skill', zorder=-
1)
  pyplot.plot(fpr, tpr, marker='.', label='MTL',  zorder=-1)
  pyplot.scatter(fpr[ix], tpr[ix], marker='o', color='black',
label='Best', zorder=2)
  # axis labels
  pyplot.xlabel('False Positive Rate')
  pyplot.ylabel('True Positive Rate')
  pyplot.legend()
  sn.set(font_scale=1.3)
  # show the plot
  pyplot.show()

  # calculate precision and recall for each threshold
  lr_precision, lr_recall, thresholds = precision_recall_curve(y_true,
lr_probs)
  # convert to f score
  fscore = (2 * lr_precision * lr_recall) / (lr_precision + lr_recall)
  ix = np.argmax(fscore)
  print('Best Threshold=%f, F-Score=%.3f' % (thresholds[ix],
fscore[ix]))
  # calculate scores
  #lr_f1, lr_auc = f1_score(y_over, y_pred), auc(lr_recall,
lr_precision)
  # summarize scores
  #print('NN: f1=%.3f auc=%.3f' % (lr_f1, lr_auc))
  # plot the roc curve for the model
  no_skill = len(y_true[y_true==1]) / len(y_true)
  pyplot.plot([0,1], [no_skill,no_skill], linestyle='--', label='No
Skill', zorder=-1)
  pyplot.plot(lr_recall, lr_precision, marker='.', label='MTL',
zorder=-1)
  pyplot.scatter(lr_recall[ix], lr_precision[ix], marker='o',
color='black', label='Best', zorder=2)
```

```
  # axis labels
  pyplot.xlabel('Recall')
  pyplot.ylabel('Precision')
  pyplot.legend()
  sn.set(font_scale=1.3)
  # show the plot
  pyplot.show()

  pred_threshold[each] = thresholds[ix]

  y_pred = [1 * (x[0]>thresholds[ix]) for x in predictions]

  matrix = confusion_matrix(y_true, y_pred)
  print("y_test value counts:\n",Counter(y_true))
  print("y_pred value counts: ", Counter(y_pred))
  print("\nConfusion matrix: ")
  print(matrix)
  cmn = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]
  sn.heatmap(cmn, annot=True)
  sn.set(font_scale=2)
  plt.ylabel('Actual')
  plt.xlabel('Predicted')
  plt.show()

  tn, fp, fn, tp = matrix.ravel()
  recall = tp/(tp+fn)
  precision = tp/(fp+tp)
  specifcity = tn/(fp+tn)
  acc = (tn+tp)/(tn+fp+fn+tp)
  auc_score = auc(lr_recall, lr_precision)
  print("Precision: ", precision)
  print("Sensitivity/recall: ", recall)
  print("Specificity: ", specifcity)
  print("Accuracy: ", acc)
  print('PR AUC: %.3f' % auc_score)

  f_score = f1_score(y_true, y_pred)
  print("f_score: ", f_score)
```

```
-----Preparing initial datasets - Train and holdout-----

Distribution of Dataset B
shape of X_trainB:  (491796, 39)
shape of y_trainB:  (491796, 3)
shape of X_testB:  (10024, 39)
shape of y_testB:  (10024, 3)

Distribution of Dataset S
shape of X_trainS:  (149071, 21)
shape of y_trainS:  (149071, 3)
```

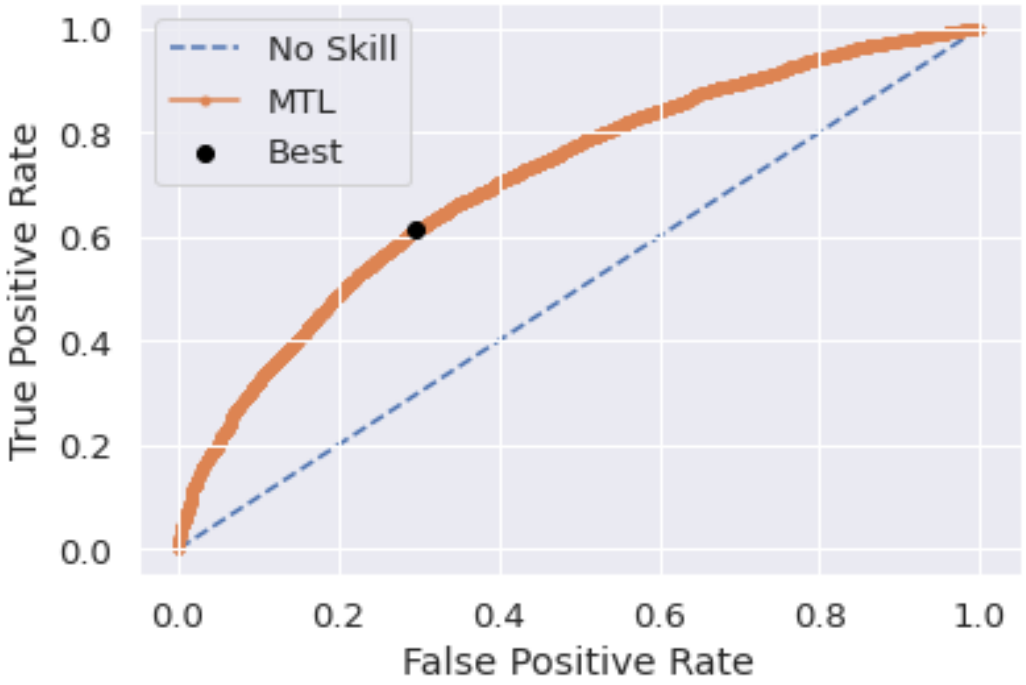
```
shape of X_testS:  (10024, 21)
shape of y_testS:  (10024, 3)

Dropping 000 rows from the Dataset A
['out'] count before SMOTE:
 000    136596
001     90581
010     83865
011     62770
101      9426
111      6289
100      2576
110      1333
Name: out, dtype: int64

Oversampling the dataset B using SMOTE-NC

Class distribution in training - Dataset B:
000    90581
001    90581
010    90581
011    90581
100    90581
101    90581
110    90581
111    90581
Name: out, dtype: int64

Preparing training data for Dataset S
Dataset B train shape:  (724648, 39)
Dataset S train shape:  (724648, 21)

Preparing training data for Dataset S
Dataset B validation shape:  (98360, 39)
Dataset S validation shape:  (98360, 21)
------------------------------------------------------------------------
--
Training for fold 1 ...

Dropping 000 rows from the Dataset A
['out'] count before SMOTE:
 000    136700
001     90447
010     83912
011     62753
101      9390
111      6313
100      2614
110      1308
Name: out, dtype: int64
```

```
Oversampling the dataset B using SMOTE-NC

Class distribution in training - Dataset B:
000    90447
001    90447
010    90447
011    90447
100    90447
101    90447
110    90447
111    90447
Name: out, dtype: int64

Preparing training data for Dataset S
Dataset B train shape:  (723576, 39)
Dataset S train shape:  (723576, 21)

Preparing training data for Dataset S
Dataset B validation shape:  (98359, 39)
Dataset S validation shape:  (98359, 21)
------------------------------------------------------------------------
--
Training for fold 2 ...

Dropping 000 rows from the Dataset A
['out'] count before SMOTE:
 000    136511
001     90528
010     83924
011     62849
101      9389
111      6363
100      2560
110      1313
Name: out, dtype: int64

Oversampling the dataset B using SMOTE-NC

Class distribution in training - Dataset B:
000    90528
001    90528
010    90528
011    90528
100    90528
101    90528
110    90528
111    90528
Name: out, dtype: int64
```

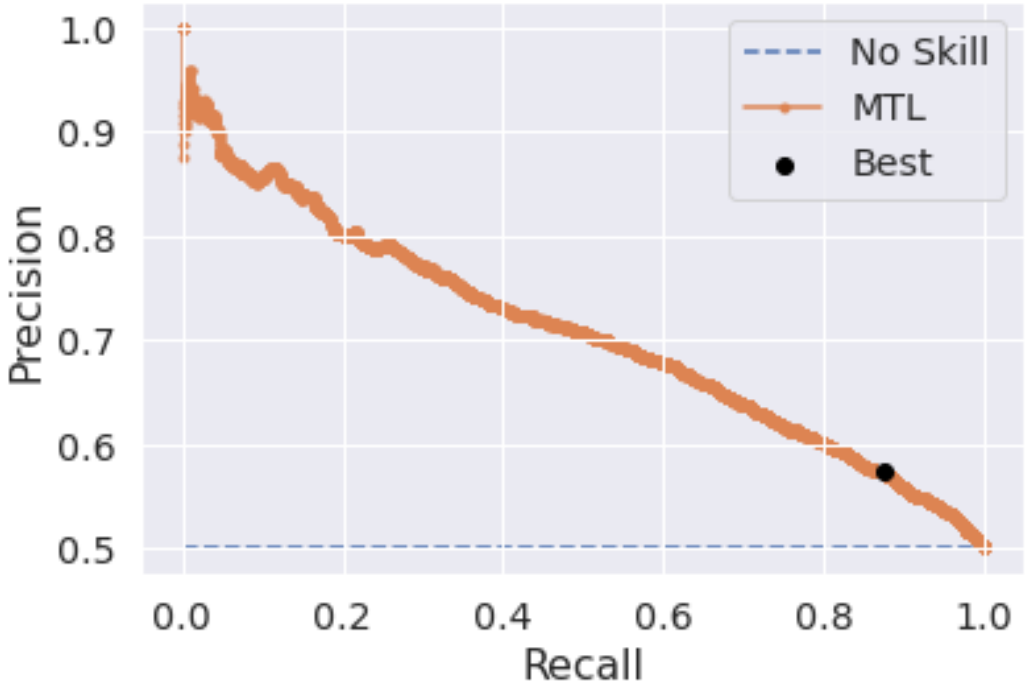
```
Preparing training data for Dataset S
Dataset B train shape:  (724224, 39)
Dataset S train shape:  (724224, 21)

Preparing training data for Dataset S
Dataset B validation shape:  (98359, 39)
Dataset S validation shape:  (98359, 21)
------------------------------------------------------------------------
--
Training for fold 3 ...

Dropping 000 rows from the Dataset A
['out'] count before SMOTE:
 000    136620
001     90621
010     83808
011     62763
101      9435
111      6319
100      2565
110      1306
Name: out, dtype: int64

Oversampling the dataset B using SMOTE-NC

Class distribution in training - Dataset B:
000    90621
001    90621
010    90621
011    90621
100    90621
101    90621
110    90621
111    90621
Name: out, dtype: int64

Preparing training data for Dataset S
Dataset B train shape:  (724968, 39)
Dataset S train shape:  (724968, 21)

Preparing training data for Dataset S
Dataset B validation shape:  (98359, 39)
Dataset S validation shape:  (98359, 21)
------------------------------------------------------------------------
--
Training for fold 4 ...

Dropping 000 rows from the Dataset A
```

```
['out'] count before SMOTE:
 000    136565
001     90687
010     83699
011     62861
101      9416
111      6268
100      2617
110      1324
Name: out, dtype: int64

Oversampling the dataset B using SMOTE-NC

Class distribution in training - Dataset B:
000    90687
001    90687
010    90687
011    90687
100    90687
101    90687
110    90687
111    90687
Name: out, dtype: int64

Preparing training data for Dataset S
Dataset B train shape:  (725496, 39)
Dataset S train shape:  (725496, 21)

Preparing training data for Dataset S
Dataset B validation shape:  (98359, 39)
Dataset S validation shape:  (98359, 21)
------------------------------------------------------------------------
--
Training for fold 5 ...

Class distribution in test - Dataset B:
011    1253
001    1253
000    1253
010    1253
101    1253
111    1253
110    1253
100    1253
Name: all, dtype: int64

Class distribution in test - Dataset S:
011    1253
001    1253
000    1253
```

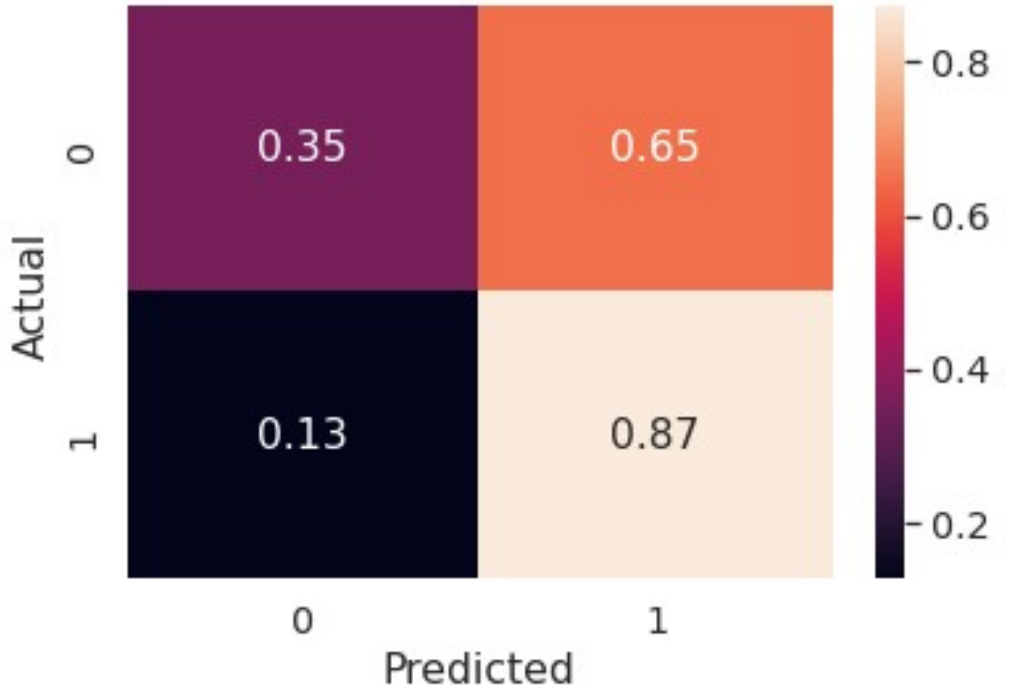
```
010    1253
101    1253
111    1253
110    1253
100    1253
Name: all, dtype: int64
Input size:  3622912

-------------------------------------------------------
diabetes
-------------------------------------------------------
Best Threshold=0.282983, G-Mean=0.658
```



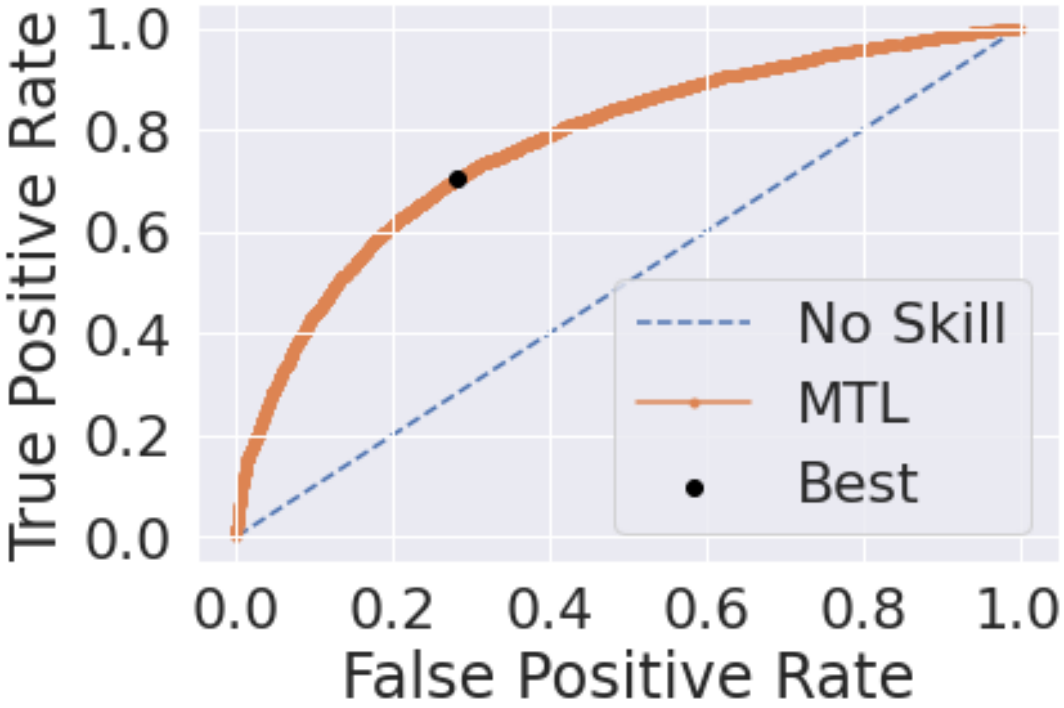

```
Best Threshold=0.085308, F-Score=0.693
```



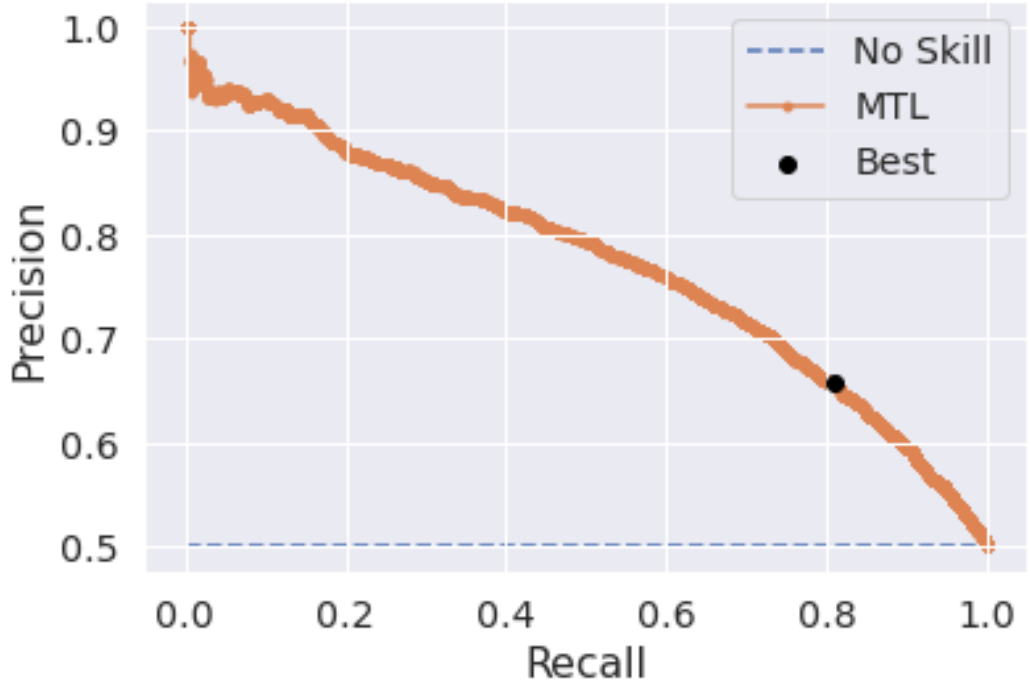

```
y_test value counts:
 Counter({0: 5012, 1: 5012})
y_pred value counts:  Counter({1: 7634, 0: 2390})

Confusion matrix:
[[1760 3252]
 [ 630 4382]]
```




```
Precision:  0.5740110034058161
Sensitivity/recall:  0.8743016759776536
Specificity:  0.35115722266560256
Accuracy:  0.6127294493216281
PR AUC: 0.707
f_score:  0.6930254625968687

-------------------------------------------------------
depression
-------------------------------------------------------
Best Threshold=0.515519, G-Mean=0.711
```

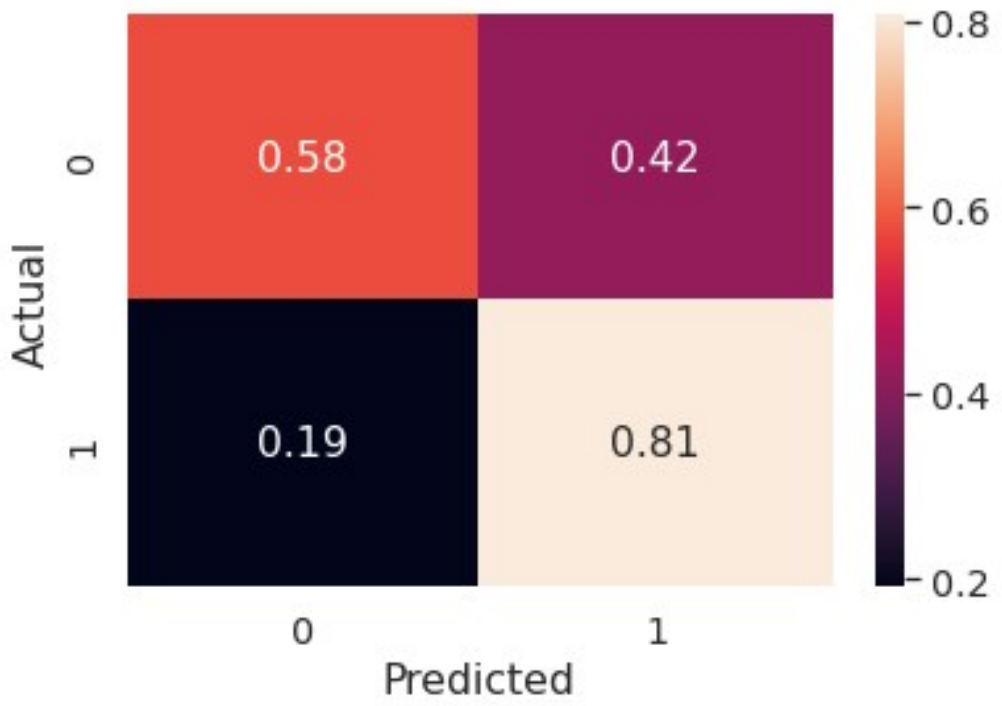



```
Best Threshold=0.388069, F-Score=0.725
```

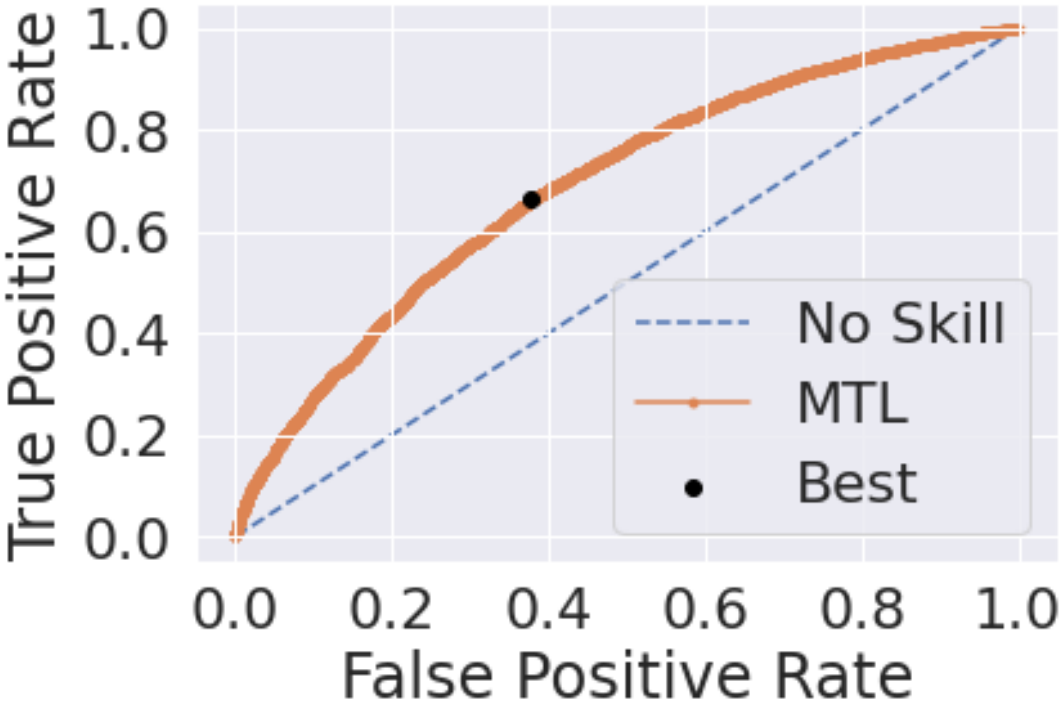



```
y_test value counts:
 Counter({1: 5012, 0: 5012})
y_pred value counts:  Counter({1: 6166, 0: 3858})
```

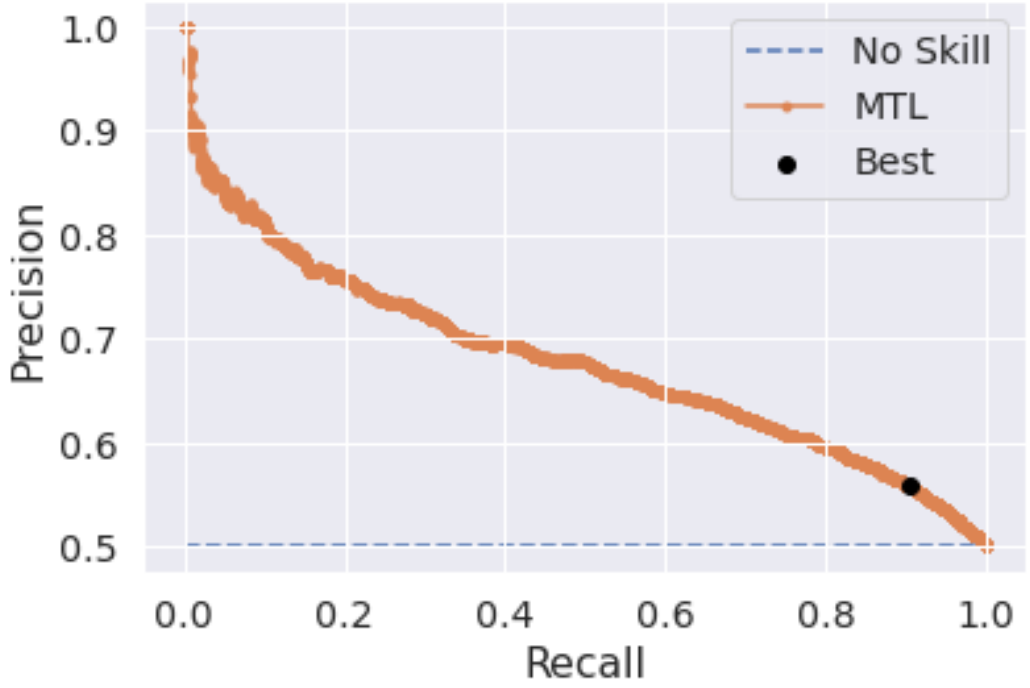
```
Confusion matrix:
[[2897 2115]
 [ 961 4051]]
```




```
Precision:  0.6569899448589037
Sensitivity/recall:  0.8082601755786113
Specificity:  0.5780127693535515
Accuracy:  0.6931364724660815
PR AUC: 0.774
f_score:  0.7248166040436572

------------------------------------------------------
cvd
------------------------------------------------------
Best Threshold=0.521185, G-Mean=0.643
```

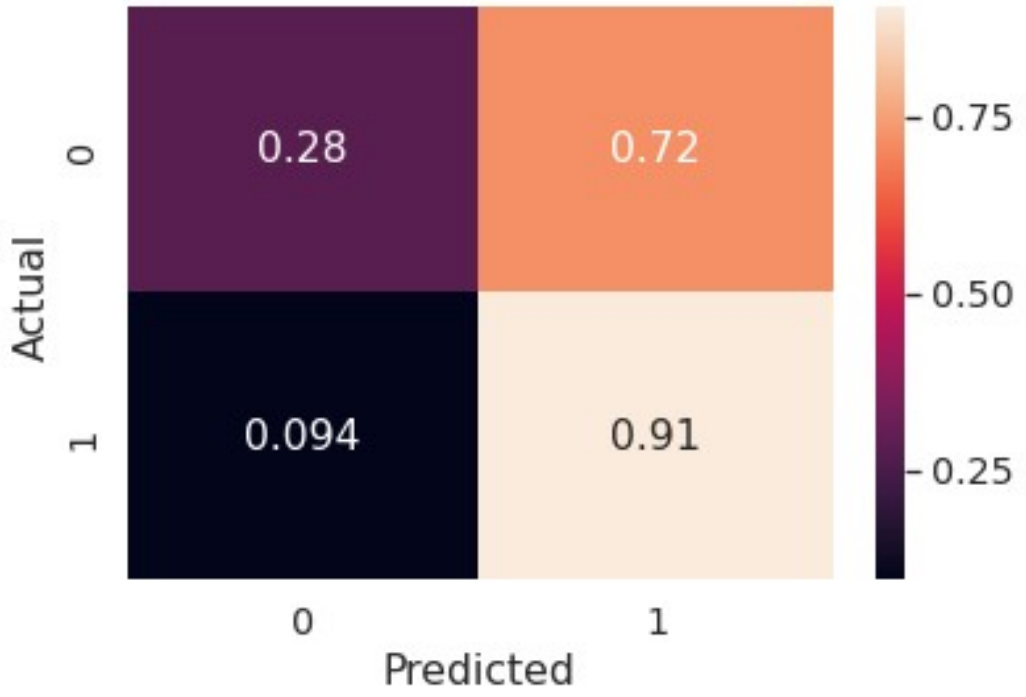



```
Best Threshold=0.322540, F-Score=0.691
```




```
y_test value counts:
 Counter({1: 5012, 0: 5012})
y_pred value counts:  Counter({1: 8130, 0: 1894})
```

```
Confusion matrix:
[[1423 3589]
 [ 471 4541]]
```




```
Precision:  0.5585485854858548
Sensitivity/recall:  0.906025538707103
Specificity:  0.28391859537110936
Accuracy:  0.5949720670391061
PR AUC: 0.678
f_score:  0.6910668087049155
```

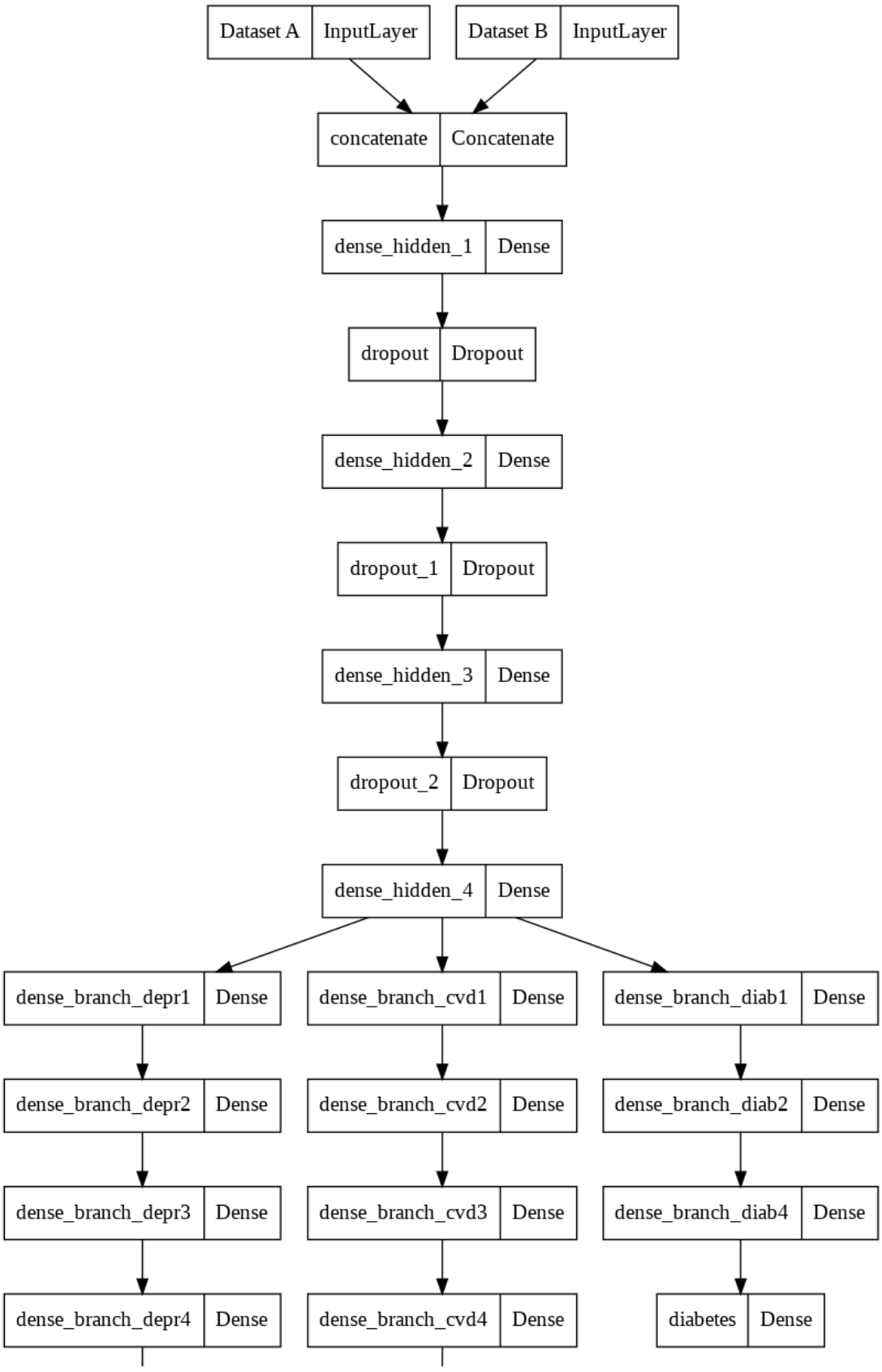
```
print("Number of layers: ", len(model.layers))
model.summary()
```

```
Number of layers:  24
Model: "model_27"
__________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==========================================================================================
 Dataset A (InputLayer)         [(None, 39)]         0           []


 Dataset B (InputLayer)         [(None, 21)]         0           []
```

```
 concatenate_5 (Concatenate)    (None, 60)           0
['Dataset A[0][0]',

'Dataset B[0][0]']


 dense_hidden_1 (Dense)         (None, 60)           3660
['concatenate_5[0][0]']


 dropout_15 (Dropout)           (None, 60)           0
['dense_hidden_1[0][0]']


 dense_hidden_2 (Dense)         (None, 55)           3355
['dropout_15[0][0]']


 dropout_16 (Dropout)           (None, 55)           0
['dense_hidden_2[0][0]']


 dense_hidden_3 (Dense)         (None, 50)           2800
['dropout_16[0][0]']


 dropout_17 (Dropout)           (None, 50)           0
['dense_hidden_3[0][0]']


 dense_hidden_4 (Dense)         (None, 45)           2295
['dropout_17[0][0]']


 dense_branch_depr1 (Dense)     (None, 40)           1840
['dense_hidden_4[0][0]']


 dense_branch_cvd1 (Dense)      (None, 40)           1840
['dense_hidden_4[0][0]']


 dense_branch_diab1 (Dense)     (None, 40)           1840
['dense_hidden_4[0][0]']
```

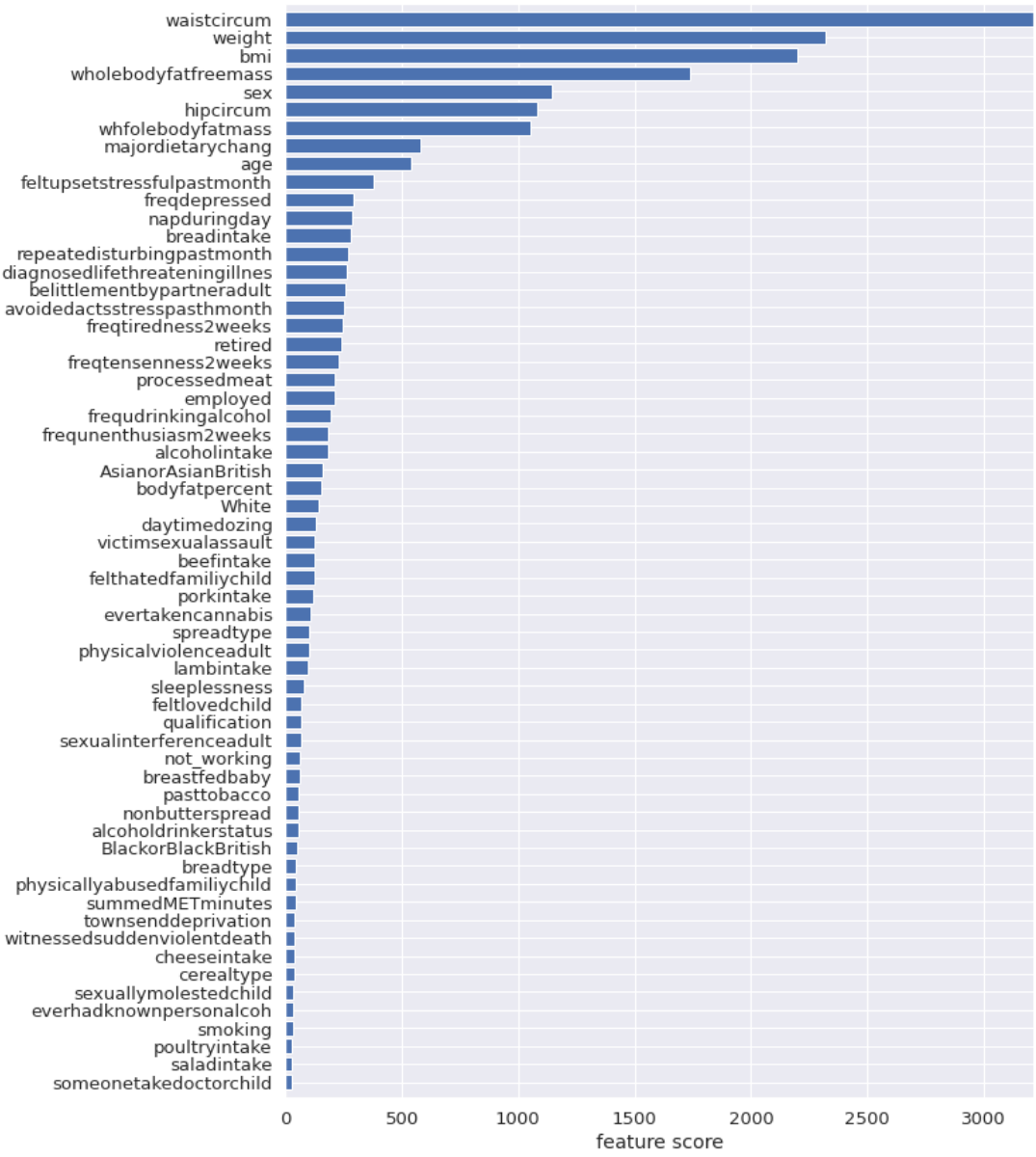
```
 dense_branch_depr2 (Dense)     (None, 35)           1435
['dense_branch_depr1[0][0]']

 dense_branch_cvd2 (Dense)      (None, 35)           1435
['dense_branch_cvd1[0][0]']

 dense_branch_diab2 (Dense)     (None, 35)           1435
['dense_branch_diab1[0][0]']

 dense_branch_depr3 (Dense)     (None, 30)           1080
['dense_branch_depr2[0][0]']

 dense_branch_cvd3 (Dense)      (None, 30)           1080
['dense_branch_cvd2[0][0]']

 dense_branch_diab4 (Dense)     (None, 20)           720
['dense_branch_diab2[0][0]']

 dense_branch_depr4 (Dense)     (None, 20)           620
['dense_branch_depr3[0][0]']

 dense_branch_cvd4 (Dense)      (None, 20)           620
['dense_branch_cvd3[0][0]']

 diabetes (Dense)               (None, 1)            21
['dense_branch_diab4[0][0]']

 depression (Dense)             (None, 1)            21
['dense_branch_depr4[0][0]']

 cvd (Dense)                    (None, 1)            21
['dense_branch_cvd4[0][0]']

==================================================================
============================
Total params: 26,118
```

```
Trainable params: 26,118
Non-trainable params: 0
_________________________________________________________________
_________________________________

tf.keras.utils.plot_model(model)
```


Dataset A
InputLayer
Dataset B
InputLayer
concatenate
Concatenate
dense_hidden_1
Dense
dropout
Dropout
dense_hidden_2
Dense
dropout_1
Dropout
dense_hidden_3
Dense
dropout_2
Dropout
dense_hidden_4
Dense
dense_branch_depr1
Dense
dense_branch_depr2
Dense
dense_branch_depr3
Dense
dense_branch_depr4
Dense
dense_branch_cvd1
Dense
dense_branch_cvd2
Dense
dense_branch_cvd3
Dense
dense_branch_cvd4
Dense
dense_branch_diab1
Dense
dense_branch_diab2
Dense
dense_branch_diab4
Dense
diabetes
Dense

Experiment 2: Diabetes and Depression

```
#Best feature selection

df = dfm.copy()
df['all'] =  df['diabetes'].astype(str) + df['depression'].astype(str)
df = df.loc[(df['all'] == '01') | (df['all'] == '10') | (df['all'] ==
'11')]
#reset the indexes of the newly created dataframe
df.reset_index(drop=True, inplace=True)

cols = df.columns.tolist()
cols.remove('diabetes')
cols.remove('depression')
cols.remove('cvd')
cols.remove('all')
df_X = df[cols]
df_Y = df[['all']]

# define feature selection
fs = SelectKBest(k=60).fit(df_X,df_Y)

X_selected = fs.transform(df_X)

# Get columns to keep and create new dataframe with those only
b_cols = fs.get_support(indices=True)
df_new = df_X.iloc[:,b_cols]
bestcols = df_new.columns.to_list()
scores = fs.scores_

li = df_X.columns.to_list()
scoresl = scores.tolist()
bestscore = []
for each in bestcols:
  bestscore.append(scoresl[li.index(each)])

res = {}
for each in bestcols:
  res[each] = bestscore[bestcols.index(each)]
ress = {k: v for k, v in sorted(res.items(), key=lambda item:
item[1])}

plt.figure(figsize=(10,15))
plt.margins(x=0, y=0.008)
plt.xlabel("feature score")
plt.barh(list(ress.keys()), list(ress.values()))
sn.set(font_scale=1.2)
plt.show()
```




```
dfB_ = dfB.copy()
dfS_ = dfS.copy()
dfB_['all'] =  dfB_['diabetes'].astype(str) +
dfB_['depression'].astype(str)
dfS_['all'] =  dfS_['diabetes'].astype(str) +
dfS_['depression'].astype(str)

dfB_ = dfB_.loc[(dfB_['all'] == '10') | (dfB_['all'] == '01') |
(dfB_['all'] == '11')]
dfS_ = dfS_.loc[(dfS_['all'] == '10') | (dfS_['all'] == '01') |
(dfB_['all'] == '11')]
```

```
#keep the columns which are in bestcols
bigcols = []
for each in dfB_.columns.to_list():
  if each in bestcols:
    bigcols.append(each)

smallcols = []
for each in dfS_.columns.to_list():
  if each in bestcols:
    smallcols.append(each)

nB_cols = len(bigcols)
nS_cols = len(smallcols)

#reset the indexes of the newly created dataframes
dfB_.reset_index(drop=True, inplace=True)
dfS_.reset_index(drop=True, inplace=True)

print("shape of dfA_: ", dfB_.shape)
print("shape of dfB_: ", dfS_.shape)


print("-----Preparing initial datasets - Train and holdout-----")
cols = dfB_.columns.tolist()
cols.remove('diabetes')
cols.remove('depression')
cols.remove('cvd')

df_XB = dfB[bigcols]
df_YB = dfB[['diabetes', 'depression']]

colsS = dfS.columns.tolist()
colsS.remove('diabetes')
colsS.remove('depression')
colsS.remove('cvd')

df_XS = dfS[smallcols]
df_YS = dfS[['diabetes', 'depression']]


#prepare train and test dataset manually from the bigger dataset
test_size  = int(df_XB.shape[0] * 0.02)
df1 = df_YB.copy()
df1['out'] = df1['diabetes'].astype(str) +
df1['depression'].astype(str)
dis_cols  = df1['out'].unique().tolist()
rowsfortestB = []
for each in dis_cols:
  tempY = df1[df1.out == each]
```

```
  rows = np.random.choice(tempY.index.values,
int(test_size/len(dis_cols)))
  rowsfortestB.append(rows)

rowsfortestB = [j for i in rowsfortestB for j in i]
X_testB = df_XB.loc[rowsfortestB]
y_testB = df_YB.loc[rowsfortestB]
X_trainB = df_XB.loc[~df_XB.index.isin(rowsfortestB)]
y_trainB= df_YB.loc[~df_YB.index.isin(rowsfortestB)]

print("\nDistribution of Dataset B")
print("shape of X_trainB: ", X_trainB.shape)
print("shape of y_trainB: ", y_trainB.shape)
print("shape of X_testB: ", X_testB.shape)
print("shape of y_testB: ", y_testB.shape)

#prepare train and test dataset manually from the small dataset
df1S = df_YS.copy()
df1S['out'] = df1S['diabetes'].astype(str)
+df1S['depression'].astype(str)

rowsfortestS = []
for each in dis_cols:
  tempY = df1S[df1S.out == each]
  rows = np.random.choice(tempY.index.values,
int(test_size/len(dis_cols)))
  rowsfortestS.append(rows)

rowsfortestS = [j for i in rowsfortestS for j in i]
X_testS = df_XS.loc[rowsfortestS]
y_testS = df_YS.loc[rowsfortestS]
X_trainS = df_XS.loc[~df_XS.index.isin(rowsfortestS)]
y_trainS = df_YS.loc[~df_YS.index.isin(rowsfortestS)]

print("\nDistribution of Dataset S")
print("shape of X_trainS: ", X_trainS.shape)
print("shape of y_trainS: ", y_trainS.shape)
print("shape of X_testS: ", X_testS.shape)
print("shape of y_testS: ", y_testS.shape)


#get indices of categorical features

#Dataset S
cols = X_trainS.columns.tolist()
categorical_featS = []
for each in cols:
    if feat_cat[each] == 'cat':
        categorical_featS.append(cols.index(each))
```

```
print(categorical_featS)

#Datset B
cols = X_trainB.columns.tolist()
categorical_featB = []
for each in cols:
    if feat_cat[each] == 'cat':
        categorical_featB.append(cols.index(each))
print(categorical_featB)


#for smaller dataset
# Standardize train and test dataset
trans = StandardScaler()
trans.fit(X_trainB)

data1 = trans.transform(X_trainB)
X_trainB = pd.DataFrame(data1)
#Apply same parameters of standardization on test as on train
data2 = trans.transform(X_testB)
X_testB = pd.DataFrame(data2)

#for smaller dataset
# Standardize train and test dataset
trans1 = StandardScaler()
trans1.fit(X_trainS)

data1 = trans1.transform(X_trainS)
X_trainS = pd.DataFrame(data1)
#Apply same parameters of standardization on test as on train
data2 = trans1.transform(X_testS)
X_testS = pd.DataFrame(data2)


# Model configuration
batch_size = 500
no_epochs = 5
verbosity = 1
num_folds = 5

# Define per-fold score containers
acc_per_fold = []
loss_per_fold = []


# Define the K-fold Cross Validator
kfold = StratifiedKFold(n_splits=num_folds, shuffle=True)
rskf = RepeatedStratifiedKFold(n_splits=num_folds, n_repeats=2,
random_state=36851234)
```

```
# K-fold Cross Validation model evaluation
fold_no = 1

inputs = X_trainB.copy()
targets = y_trainB.copy()

inputsize = 0

# Define the layers of the model
input_A = Input(shape = (nB_cols,), name = 'Dataset A')
input_B = Input(shape = (nS_cols,), name = 'Dataset B')
out = Concatenate()([input_A, input_B])
dense_hidden_1 = Dense(55, activation = 'relu', name =
'dense_hidden_1')(out)
dense_hidden_1_d = Dropout(0.5)(dense_hidden_1)
dense_hidden_2 = Dense(50, activation = 'relu', name =
'dense_hidden_2')(dense_hidden_1_d)
dense_hidden_2_d = Dropout(0.25)(dense_hidden_2)
dense_hidden_3 = Dense(45, activation = 'relu', name =
'dense_hidden_3')(dense_hidden_2_d)
dense_hidden_3_d = Dropout(0.1)(dense_hidden_3)
dense_hidden_4 = Dense(40, activation = 'relu', name =
'dense_hidden_4')(dense_hidden_3_d)


dense_branch_diab1 = Dense(35, activation = 'relu', name =
'dense_branch_diab1')(dense_hidden_4)
#dense_hidden_diab1_d = Dropout(0.25)(dense_branch_diab1)
dense_branch_diab2 = Dense(30, activation = 'relu', name =
'dense_branch_diab2')(dense_branch_diab1)
dense_branch_diab3 = Dense(25, activation = 'relu', name =
'dense_branch_diab3')(dense_branch_diab2)
dense_branch_diab4 = Dense(20, activation = 'relu', name =
'dense_branch_diab4')(dense_branch_diab2)
diabetes = Dense(1, activation = 'sigmoid', name = 'diabetes')
(dense_branch_diab4)


dense_branch_depr1 = Dense(35, activation = 'relu', name =
'dense_branch_depr1')(dense_hidden_4)
#dense_hidden_depr1_d = Dropout(0.1)(dense_branch_depr1)
dense_branch_depr2 = Dense(30, activation = 'relu', name =
'dense_branch_depr2')(dense_branch_depr1)
dense_branch_depr3 = Dense(25, activation = 'relu', name =
'dense_branch_depr3')(dense_branch_depr2)
dense_branch_depr4 = Dense(20, activation = 'relu', name =
'dense_branch_depr4')(dense_branch_depr3)
depression = Dense(1, activation = 'sigmoid', name ='depression')
```

```
(dense_branch_depr4)

dense_branch_cvd1 = Dense(35, activation = 'relu', name =
'dense_branch_cvd1')(dense_hidden_4)
#dense_hidden_cvd1_d = Dropout(0.25)(dense_branch_cvd1)
dense_branch_cvd2 = Dense(30, activation = 'relu', name =
'dense_branch_cvd2')(dense_branch_cvd1)
dense_branch_cvd3 = Dense(25, activation = 'relu', name =
'dense_branch_cvd3')(dense_branch_cvd2)
dense_branch_cvd4 = Dense(20, activation = 'relu', name =
'dense_branch_cvd4')(dense_branch_cvd3)
cvd = Dense(1, activation = 'sigmoid', name ='cvd')(dense_branch_cvd4)

for train, test in kfold.split(inputs, targets['diabetes']):

  trainX = inputs.iloc[train]
  trainy = targets.iloc[train]
  testXB = inputs.iloc[test]
  testyB = targets.iloc[test]
  #print("shape of trainX: ", trainX.shape)
  #print("trainy orig unique: \n", trainy.value_counts())

  # split the small dataset into train and validation
  X_train_s, X_tests, y_train_s, y_tests = train_test_split(X_trainS,
y_trainS, test_size=0.20,  stratify=y_trainS['diabetes'])


  #merge the target columns for undersampling
  df1 = trainy.copy()
  df1['out'] = df1['diabetes'].astype(str)
+df1['depression'].astype(str)

  #drop some 00 rows
  orig_count = df1['out'].value_counts().to_dict()
  orig_count['00'] = 0
  all_values = orig_count.values()
  orig_count['00'] = int(max(all_values))
  #print("orig_count after: ", orig_count)
  # define the undersampling method
  undersample = NearMiss(version=1, n_neighbors=3, sampling_strategy =
orig_count)
  trainy = df1['out']
  df_US_X, df_US_y = undersample.fit_resample(trainX, trainy)

  print("\nOversampling the dataset B using SMOTE-NC")
  #oversample the dataset
  samp_count = df1['out'].value_counts().to_dict()
  print("Class distribution before smote: ", samp_count)
  #samp_count['100'] = int(samp_count['000']/2)
  #samp_count['110'] = int(samp_count['000']/2)
```

```
  oversample = SMOTENC(categorical_features=categorical_featB)
  df_OS_X, df_OS_y = oversample.fit_resample(df_US_X, df_US_y)


  #split the 'out column to 3 target diseases
  a = pd.DataFrame(df_OS_y)
  a.columns = ['out']
  print("\nClass distribution in training - Dataset B:")
  print(a['out'].value_counts())
  #print("['out'] count after SMOTE: \n",a['out'].value_counts())
  a['diabetes'] = a['out'].str[:1]
  a['depression'] = a['out'].str[1:2]

  #convert datatype of target columns to int
  a['diabetes'] = a['diabetes'].astype(str).astype(int)
  a['depression'] = a['depression'].astype(str).astype(int)
  a.drop(['out'], axis = 1)
  trainXB = df_OS_X.copy()
  trainyB = a.copy()



  #create input_S dataset for training
  print("\nPreparing training data for Dataset S")
  m = trainXB.shape[0]
  yb = trainyB.copy()
  yb['all'] = yb['diabetes'].astype(str) +yb['depression'].astype(str)
  #print("y counts for train: \n", ys['all'].value_counts())
  yb_count = yb['all'].value_counts().to_dict()
  #print("yb_count: ", yb_count)
  S_y = y_train_s.copy()
  S_y['all'] = S_y['diabetes'].astype(str)
+S_y['depression'].astype(str)
  S_y_count = S_y['all'].value_counts().to_dict()
  #print("B_y['all'] counts: \n", B_y_count)

  colmov = S_y[['diabetes', 'depression']]
  #print("shape of X_train_s: ", X_train_s.shape)
  #print("shape of S_y: ", S_y.shape)
  X_train_s.reset_index(drop=True, inplace=True)
  colmov.reset_index(drop=True, inplace=True)
  df_XSn = pd.concat([X_train_s, colmov], axis=1, join="inner")
  #print("shape of df_XSn: ", df_XSn.shape)
  #print("shape of B_y: ", B_y.shape)

  S_y_countNew = S_y_count.copy()
  for each in S_y_count:
    if S_y_count[each] > yb_count[each]:
      S_y_countNew[each] = yb_count[each]
```

```
  #some classes in S_y have more samples than in ys, drop the extra
samples
  undersample = NearMiss(version=1, n_neighbors=3, sampling_strategy =
S_y_countNew)
  df_S_X, df_S_y = undersample.fit_resample(df_XSn, S_y['all'])

  ros = RandomOverSampler(sampling_strategy = yb_count)

  X_res, y_res = ros.fit_resample(df_S_X, df_S_y)
  X_res = pd.DataFrame(X_res, columns = df_XSn.columns)
  y_res = pd.DataFrame(y_res)
  y_res.columns = ['all']

  #print("X_res shape: ", X_res.shape)
  #print("y_res shape: ", y_res.shape)
  y_res['newind'] = 0
  for each in yb_count:
    indexes = yb.loc[yb['all']==each].index.tolist()
    y_res.loc[y_res['all'] == each, 'newind'] = indexes
  X_res = pd.concat([X_res, y_res['newind']], axis=1, join="inner")
  X_res = X_res.sort_values('newind')
  X_res.reset_index(drop=True, inplace=True)
  X_res = X_res.drop(['diabetes', 'depression','newind'], axis = 1)
  trainXS = X_res.copy()

  print("Dataset B train shape: ", trainXB.shape)
  print("Dataset S train shape: ", trainXS.shape)

  #create input_S dataset for validation
  print("\nPreparing training data for Dataset S")
  m = testXB.shape[0]
  yb = testyB.copy()
  yb['all'] = yb['diabetes'].astype(str) +yb['depression'].astype(str)
  #print("y counts for train: \n", ys['all'].value_counts())
  yb_count = yb['all'].value_counts().to_dict()
  #print("yb_count: ", yb_count)
  S_y = y_tests.copy()
  S_y['all'] = S_y['diabetes'].astype(str)
+S_y['depression'].astype(str)
  S_y_count = S_y['all'].value_counts().to_dict()
  #print("B_y['all'] counts: \n", B_y_count)

  colmov = S_y[['diabetes', 'depression']]
  #print("shape of X_train_s: ", X_train_s.shape)
  #print("shape of S_y: ", S_y.shape)
  X_tests.reset_index(drop=True, inplace=True)
  colmov.reset_index(drop=True, inplace=True)
  df_XSn = pd.concat([X_tests, colmov], axis=1, join="inner")
  #print("shape of df_XSn: ", df_XSn.shape)
```

```
  #print("shape of B_y: ", B_y.shape)

  S_y_countNew = S_y_count.copy()
  for each in S_y_count:
    if S_y_count[each] > yb_count[each]:
      S_y_countNew[each] = yb_count[each]

  #some classes in S_y have more samples than in ys, drop the extra
samples
  undersample = NearMiss(version=1, n_neighbors=3, sampling_strategy =
S_y_countNew)
  df_S_X, df_S_y = undersample.fit_resample(df_XSn, S_y['all'])

  ros = RandomOverSampler(sampling_strategy = yb_count)

  X_res, y_res = ros.fit_resample(df_S_X, df_S_y)
  X_res = pd.DataFrame(X_res, columns = df_XSn.columns)
  y_res = pd.DataFrame(y_res)
  y_res.columns = ['all']

  #print("X_res shape: ", X_res.shape)
  #print("y_res shape: ", y_res.shape)
  y_res['newind'] = 0
  for each in yb_count:
    indexes = yb.loc[yb['all']==each].index.tolist()
    y_res.loc[y_res['all'] == each, 'newind'] = indexes
  X_res = pd.concat([X_res, y_res['newind']], axis=1, join="inner")
  X_res = X_res.sort_values('newind')
  X_res.reset_index(drop=True, inplace=True)
  X_res = X_res.drop(['diabetes', 'depression','newind'], axis = 1)
  testXS = X_res.copy()


  print("Dataset B validation shape: ", testXB.shape)
  print("Dataset S validation shape: ", testXS.shape)

  model = tf.keras.models.Model([input_A, input_B], [diabetes,
depression])

  initial_learning_rate = 0.1
  lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
      initial_learning_rate,
      decay_steps=10000,
      decay_rate=0.96,
      staircase=True)
  optimizersgd = tf.keras.optimizers.SGD(lr=0.0001, momentum=0.9)
  optimizeradam = tf.keras.optimizers.Adam(lr=0.0001)

  model.compile(
```

```
      loss={
          'diabetes': 'binary_crossentropy',
          'depression': 'binary_crossentropy'
      },
      optimizer = optimizeradam,
      metrics = ['accuracy']
  )

  # Generate a print

print('------------------------------------------------------------------
---------')
  print(f'Training for fold {fold_no} ...')
  #print("len(df_OS_X): ", len(df_OS_X))
  #print("len(testX): ", len(testX))

  inputsize += trainXB.shape[0]

  # Fit data to model
  history = model.fit([trainXB, trainXS],
[trainyB['diabetes'],trainyB['depression']],
            validation_data = ([testXB, testXS], [testyB['diabetes'],
testyB['depression']]),
            batch_size=batch_size,
            epochs=no_epochs,
            verbose=0)

  # Generate generalization metrics
  scores = model.evaluate([testXB, testXS], [testyB['diabetes'],
testyB['depression']], verbose=0)

#  print(f'Score for fold {fold_no}: {model.metrics_names[0]} of
{scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
# acc_per_fold.append(scores[1] * 100)
#  loss_per_fold.append(scores[0])

  # Increase fold number
  fold_no = fold_no + 1


#sort both test datasets to align target disease columns
y_testB.reset_index(drop=True, inplace=True)
X_testB.reset_index(drop=True, inplace=True)
y_testB['all'] = y_testB['diabetes'].astype(str)
+y_testB['depression'].astype(str)

#create a dataframe for target variables of test B
all_testXB = pd.concat([X_testB, y_testB['all']], axis=1,
join="inner")
```

```
all_testXB.sort_values(by=['all'])
a = pd.DataFrame(columns=['all'])
a['all'] = all_testXB['all'].copy()
all_testXB = all_testXB.drop(['all'], axis = 1)
print("\nClass distribution in test - Dataset B:")
print(a['all'].value_counts())
#print("['out'] count after SMOTE: \n",a['out'].value_counts())
a['diabetes'] = a['all'].str[:1]
a['depression'] = a['all'].str[1:2]
all_testyB = a.drop('all', axis=1)
#convert datatype of target columns to int
all_testyB['diabetes'] =
all_testyB['diabetes'].astype(str).astype(int)
all_testyB['depression'] =
all_testyB['depression'].astype(str).astype(int)


y_testS.reset_index(drop=True, inplace=True)
X_testS.reset_index(drop=True, inplace=True)
y_testS['all'] = y_testS['diabetes'].astype(str)
+y_testS['depression'].astype(str)
#create a dataframe for target variables of test S
all_testXS = pd.concat([X_testS, y_testS['all']], axis=1,
join="inner")
all_testXS.sort_values(by=['all'])
a = pd.DataFrame(columns=['all'])
a['all'] = all_testXS['all'].copy()
all_testXS = all_testXS.drop(['all'], axis = 1)
print("\nClass distribution in test - Dataset S:")
print(a['all'].value_counts())
#print("['out'] count after SMOTE: \n",a['out'].value_counts())
a['diabetes'] = a['all'].str[:1]
a['depression'] = a['all'].str[1:2]
all_testyS = a.drop('all', axis=1)
#convert datatype of target columns to int
all_testyS['diabetes'] =
all_testyS['diabetes'].astype(str).astype(int)
all_testyS['depression'] =
all_testyS['depression'].astype(str).astype(int)

print("input size: ", inputsize)

preds = model.predict([all_testXB, all_testXS])
# Metrics for each target disease
target_v = ['diabetes', 'depression']

pred_threshold = {}

for each in target_v:
  print("\n-------------------------------------------------------")
```

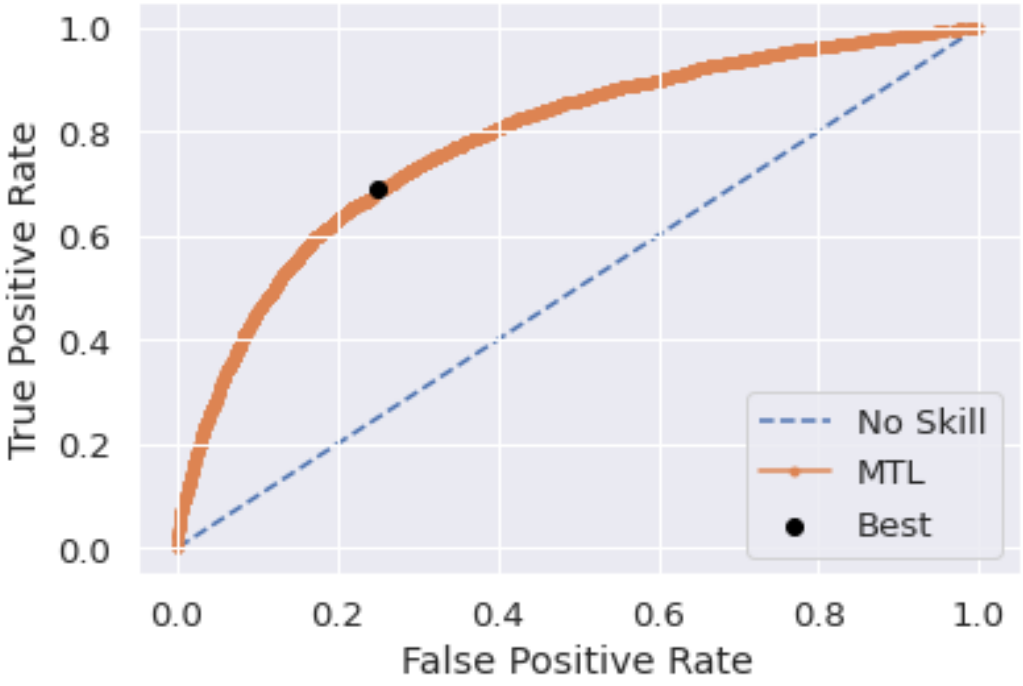
```
  print(each)
  print("-----------------------------------------------------------")

  #preds = model.predict(X_test)
  y_true = all_testyB[each]
  predictions = preds[target_v.index(each)]

  lr_probs = predictions

  fpr, tpr, thresholds = roc_curve(y_true, lr_probs)
  # calculate the g-mean for each threshold
  gmeans = np.sqrt(tpr * (1-fpr))
  # locate the index of the largest g-mean
  ix = np.argmax(gmeans)
  print('Best Threshold=%f, G-Mean=%.3f' % (thresholds[ix],
gmeans[ix]))
  # plot the roc curve for the model
  pyplot.plot([0,1], [0,1], linestyle='--', label='No Skill', zorder=-
1)
  pyplot.plot(fpr, tpr, marker='.', label='MTL',  zorder=-1)
  pyplot.scatter(fpr[ix], tpr[ix], marker='o', color='black',
label='Best', zorder=2)
  # axis labels
  pyplot.xlabel('False Positive Rate')
  pyplot.ylabel('True Positive Rate')
  pyplot.legend()
  #sn.set(font_scale=1.2)
  # show the plot
  pyplot.show()

  # calculate precision and recall for each threshold
  lr_precision, lr_recall, thresholds = precision_recall_curve(y_true,
lr_probs)
  # convert to f score
  fscore = (2 * lr_precision * lr_recall) / (lr_precision + lr_recall)
  ix = np.argmax(fscore)
  print('Best Threshold=%f, F-Score=%.3f' % (thresholds[ix],
fscore[ix]))
  # calculate scores
  #lr_f1, lr_auc = f1_score(y_over, y_pred), auc(lr_recall,
lr_precision)
  # summarize scores
  #print('NN: f1=%.3f auc=%.3f' % (lr_f1, lr_auc))
  # plot the roc curve for the model
  no_skill = len(y_true[y_true==1]) / len(y_true)
  pyplot.plot([0,1], [no_skill,no_skill], linestyle='--', label='No
Skill', zorder=-1)
  pyplot.plot(lr_recall, lr_precision, marker='.', label='MTL',
zorder=-1)
  pyplot.scatter(lr_recall[ix], lr_precision[ix], marker='o',
```

```
color='black', label='Best', zorder=2)
  # axis labels
  pyplot.xlabel('Recall')
  pyplot.ylabel('Precision')
  pyplot.legend()
  #sn.set(font_scale=1.2)
  # show the plot
  pyplot.show()

  pred_threshold[each] = thresholds[ix]

  y_pred = [1 * (x[0]>thresholds[ix]) for x in predictions]

  matrix = confusion_matrix(y_true, y_pred)
  print("y_test value counts:\n",Counter(y_true))
  print("y_pred value counts: ", Counter(y_pred))
  print("\nConfusion matrix: ")
  print(matrix)
  cmn = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]
  sn.heatmap(cmn, annot=True)
  sn.set(font_scale=2)
  plt.ylabel('Actual')
  plt.xlabel('Predicted')
  plt.show()

  tn, fp, fn, tp = matrix.ravel()
  recall = tp/(tp+fn)
  precision = tp/(fp+tp)
  specifcity = tn/(fp+tn)
  acc = (tn+tp)/(tn+fp+fn+tp)
  auc_score = auc(lr_recall, lr_precision)
  print("Precision: ", precision)
  print("Sensitivity/recall: ", recall)
  print("Specificity: ", specifcity)
  print("Accuracy: ", acc)
  print('PR AUC: %.3f' % auc_score)

  f_score = f1_score(y_true, y_pred)
  print("f_score: ", f_score)


shape of dfA_:  (214778, 71)
shape of dfB_:  (76788, 27)
-----Preparing initial datasets - Train and holdout-----

Distribution of Dataset B
shape of X_trainB:  (491662, 43)
shape of y_trainB:  (491662, 2)
shape of X_testB:  (10024, 43)
shape of y_testB:  (10024, 2)
```

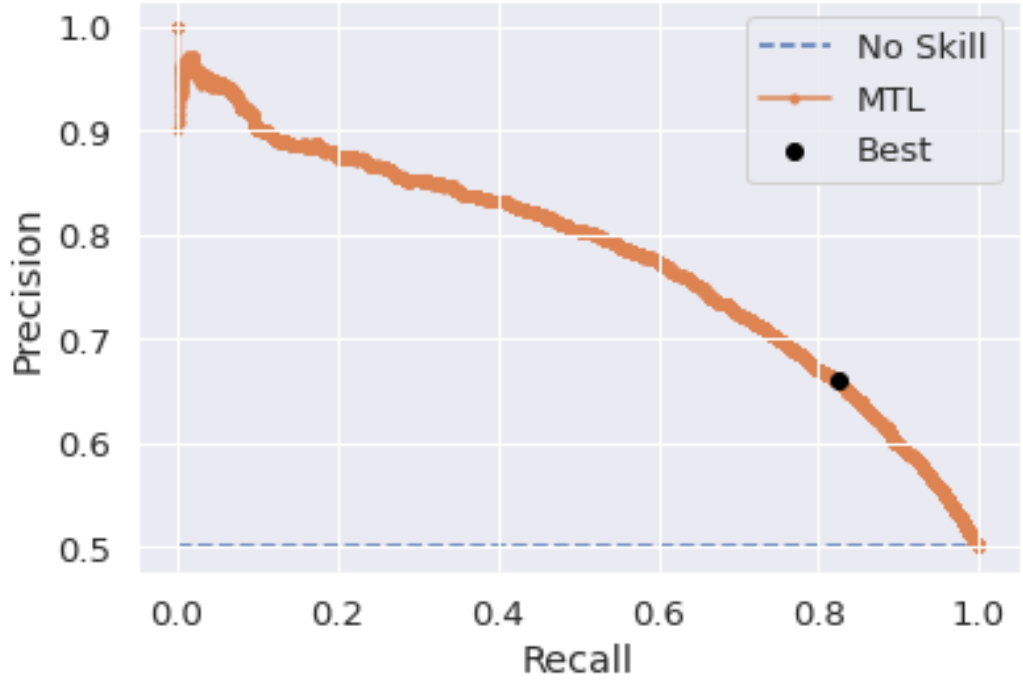
```
Distribution of Dataset S
shape of X_trainS:  (148837, 17)
shape of y_trainS:  (148837, 2)
shape of X_testS:  (10024, 17)
shape of y_testS:  (10024, 2)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
[0, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22,
23, 24, 25, 26, 27, 36, 37, 38, 39, 40, 41, 42]

Oversampling the dataset B using SMOTE-NC
Class distribution before smote:  {'00': 226926, '01': 146884, '10':
11956, '11': 7563}

Class distribution in training - Dataset B:
00    146884
01    146884
10    146884
11    146884
Name: out, dtype: int64

Preparing training data for Dataset S
Dataset B train shape:  (587536, 43)
Dataset S train shape:  (587536, 17)

Preparing training data for Dataset S
Dataset B validation shape:  (98333, 43)
Dataset S validation shape:  (98333, 17)
------------------------------------------------------------------------
--
Training for fold 1 ...

Oversampling the dataset B using SMOTE-NC
Class distribution before smote:  {'00': 227461, '01': 146349, '10':
11933, '11': 7586}

Class distribution in training - Dataset B:
00    146349
01    146349
10    146349
11    146349
Name: out, dtype: int64

Preparing training data for Dataset S
Dataset B train shape:  (585396, 43)
Dataset S train shape:  (585396, 17)

Preparing training data for Dataset S
Dataset B validation shape:  (98333, 43)
```

```
Dataset S validation shape:  (98333, 17)
------------------------------------------------------------------------
--
Training for fold 2 ...

Oversampling the dataset B using SMOTE-NC
Class distribution before smote:  {'00': 227022, '01': 146788, '10':
11953, '11': 7567}

Class distribution in training - Dataset B:
00    146788
01    146788
10    146788
11    146788
Name: out, dtype: int64

Preparing training data for Dataset S
Dataset B train shape:  (587152, 43)
Dataset S train shape:  (587152, 17)

Preparing training data for Dataset S
Dataset B validation shape:  (98332, 43)
Dataset S validation shape:  (98332, 17)
------------------------------------------------------------------------
--
Training for fold 3 ...

Oversampling the dataset B using SMOTE-NC
Class distribution before smote:  {'00': 227406, '01': 146405, '10':
11989, '11': 7530}

Class distribution in training - Dataset B:
00    146405
01    146405
10    146405
11    146405
Name: out, dtype: int64

Preparing training data for Dataset S
Dataset B train shape:  (585620, 43)
Dataset S train shape:  (585620, 17)

Preparing training data for Dataset S
Dataset B validation shape:  (98332, 43)
Dataset S validation shape:  (98332, 17)
------------------------------------------------------------------------
--
Training for fold 4 ...
```

```
Oversampling the dataset B using SMOTE-NC
Class distribution before smote:  {'00': 227061, '01': 146750, '10':
11941, '11': 7578}

Class distribution in training - Dataset B:
00    146750
01    146750
10    146750
11    146750
Name: out, dtype: int64

Preparing training data for Dataset S
Dataset B train shape:  (587000, 43)
Dataset S train shape:  (587000, 17)

Preparing training data for Dataset S
Dataset B validation shape:  (98332, 43)
Dataset S validation shape:  (98332, 17)
------------------------------------------------------------------------
--
Training for fold 5 ...

Class distribution in test - Dataset B:
01    2506
00    2506
10    2506
11    2506
Name: all, dtype: int64

Class distribution in test - Dataset S:
01    2506
00    2506
10    2506
11    2506
Name: all, dtype: int64
input size:  2932704

------------------------------------------------------
diabetes
------------------------------------------------------
Best Threshold=0.309637, G-Mean=0.718
```

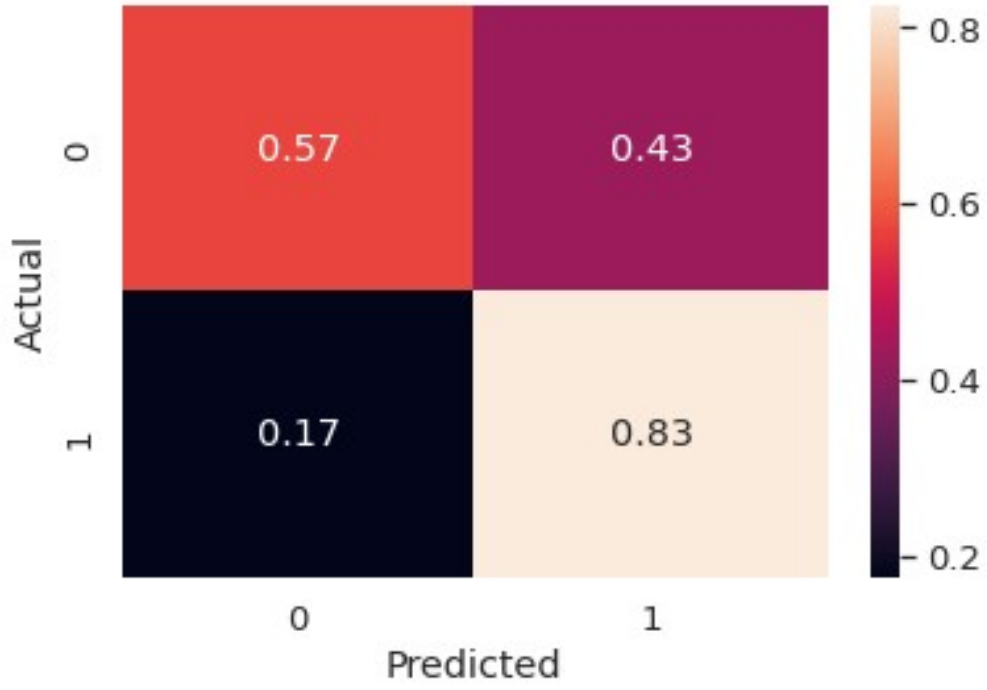



```
Best Threshold=0.170369, F-Score=0.733
```

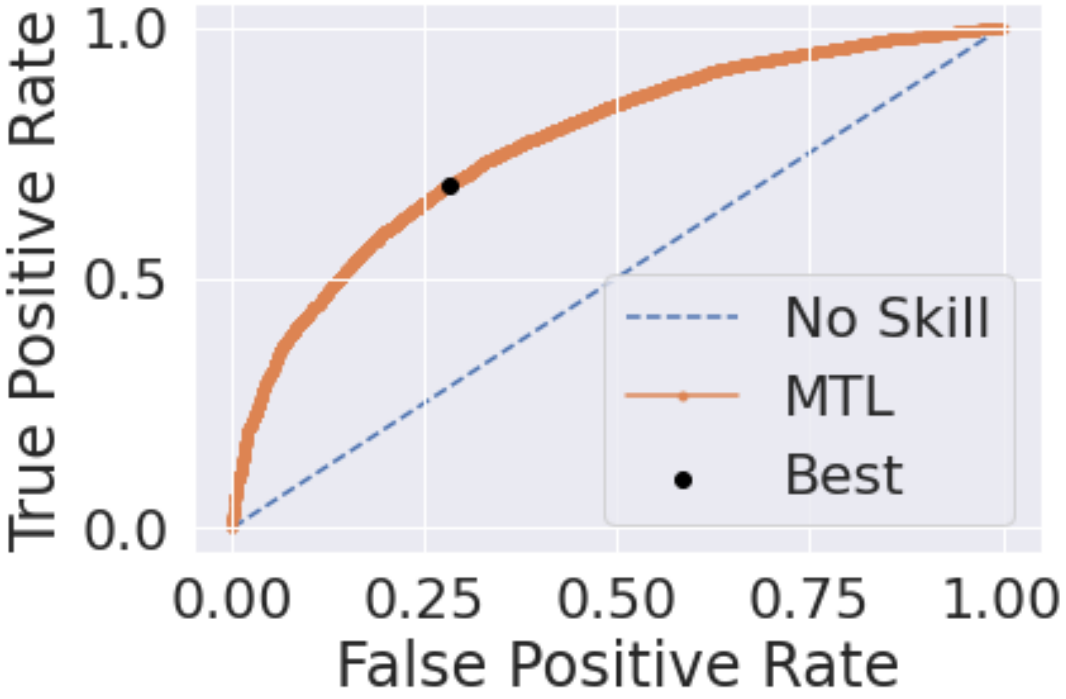



```
y_test value counts:
 Counter({0: 5012, 1: 5012})
y_pred value counts:  Counter({1: 6274, 0: 3750})
```

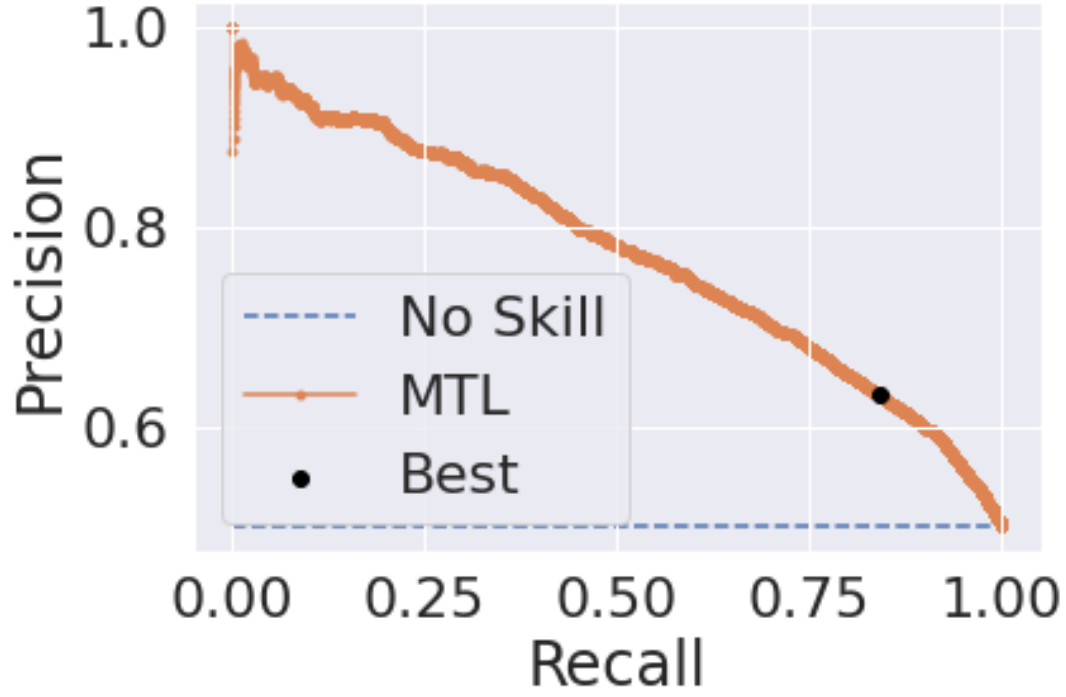
```
Confusion matrix:
[[2874 2138]
 [ 876 4136]]
```




```
Precision:  0.6592285623206886
Sensitivity/recall:  0.825219473264166
Specificity:  0.5734237829209896
Accuracy:  0.6993216280925778
PR AUC: 0.779
f_score:  0.7329434697855751

------------------------------------------------------
depression
------------------------------------------------------
Best Threshold=0.521249, G-Mean=0.703
```

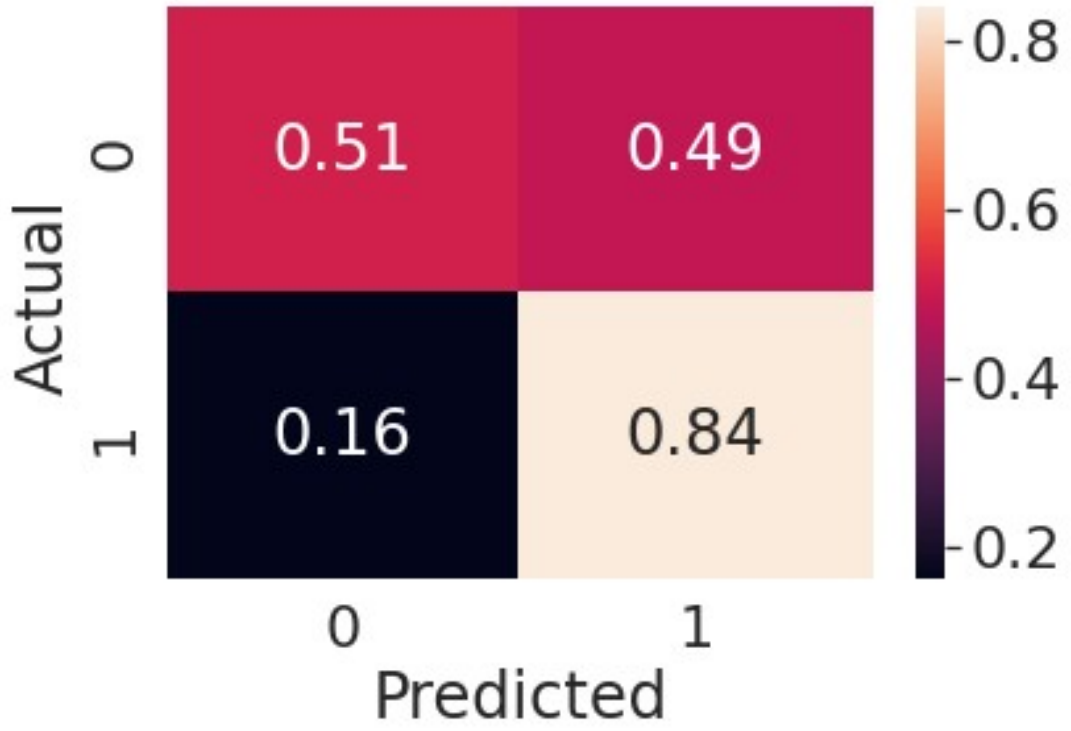



```
Best Threshold=0.347369, F-Score=0.722
```




```
y_test value counts:
 Counter({1: 5012, 0: 5012})
y_pred value counts:  Counter({1: 6641, 0: 3383})
```

```
Confusion matrix:
[[2580 2432]
 [ 803 4209]]
```



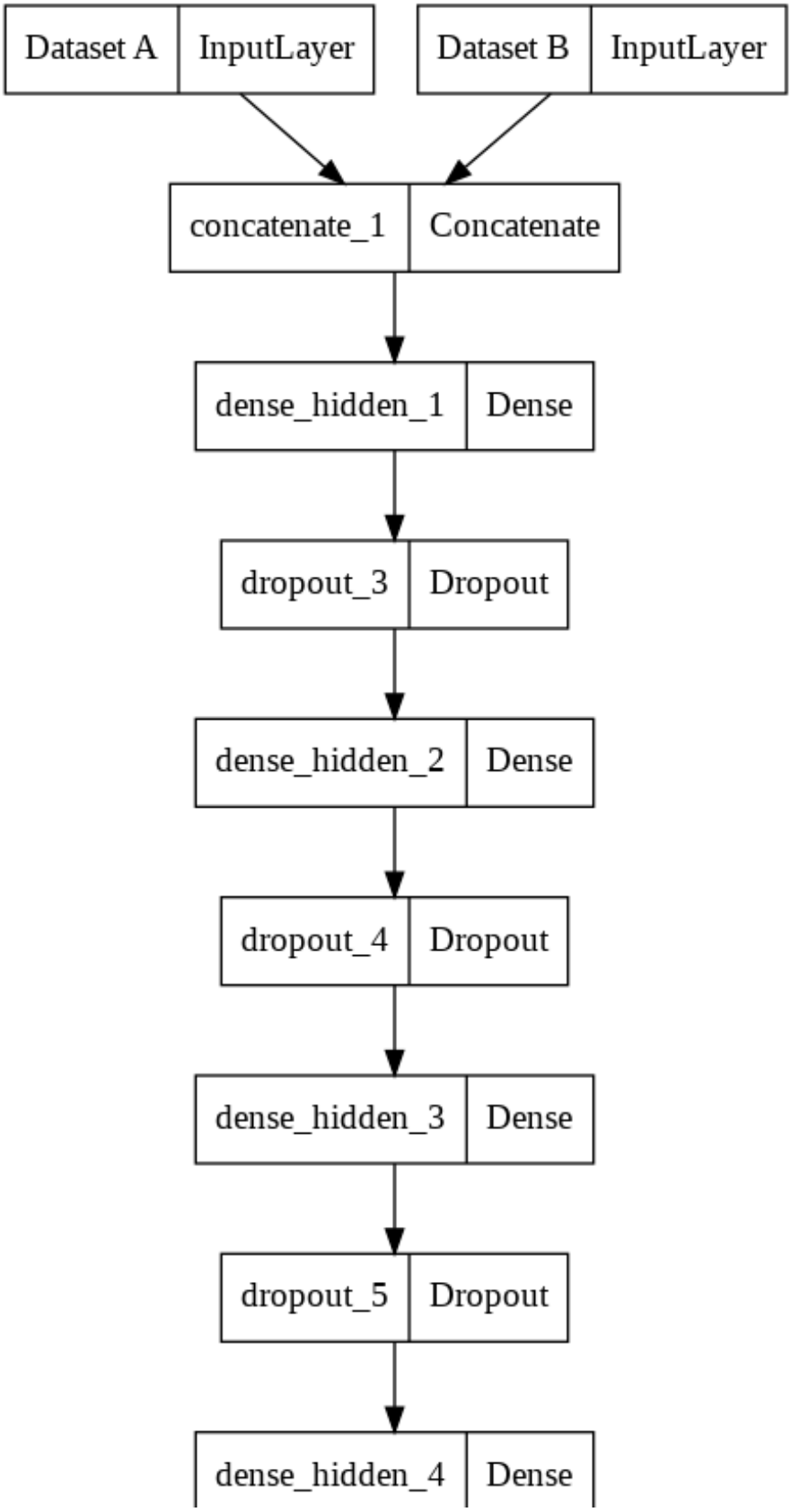

```
Precision:  0.6337900918536366
Sensitivity/recall:  0.8397845171588189
Specificity:  0.5147645650438947
Accuracy:  0.6772745411013568
PR AUC: 0.773
f_score:  0.7223890843559599
```

```
print("Number of layers: ", len(model.layers))
model.summary()
```

```
Number of layers:  19
Model: "model_22"
_____________________________________________________________________________
_____
 Layer (type)                    Output Shape         Param #
Connected to
==============================================================================
============================
 Dataset A (InputLayer)          [(None, 43)]         0           []
```

```
 Dataset B (InputLayer)         [(None, 17)]         0           []



 concatenate_4 (Concatenate)    (None, 60)           0
['Dataset A[0][0]',

'Dataset B[0][0]']


 dense_hidden_1 (Dense)         (None, 55)           3355
['concatenate_4[0][0]']


 dropout_12 (Dropout)           (None, 55)           0
['dense_hidden_1[0][0]']


 dense_hidden_2 (Dense)         (None, 50)           2800
['dropout_12[0][0]']


 dropout_13 (Dropout)           (None, 50)           0
['dense_hidden_2[0][0]']


 dense_hidden_3 (Dense)         (None, 45)           2295
['dropout_13[0][0]']


 dropout_14 (Dropout)           (None, 45)           0
['dense_hidden_3[0][0]']


 dense_hidden_4 (Dense)         (None, 40)           1840
['dropout_14[0][0]']


 dense_branch_depr1 (Dense)     (None, 35)           1435
['dense_hidden_4[0][0]']


 dense_branch_diab1 (Dense)     (None, 35)           1435
['dense_hidden_4[0][0]']


 dense_branch_depr2 (Dense)     (None, 30)           1080
['dense_branch_depr1[0][0]']
```

```
 dense_branch_diab2 (Dense)     (None, 30)           1080
['dense_branch_diab1[0][0]']


 dense_branch_depr3 (Dense)     (None, 25)           775
['dense_branch_depr2[0][0]']


 dense_branch_diab4 (Dense)     (None, 20)           620
['dense_branch_diab2[0][0]']


 dense_branch_depr4 (Dense)     (None, 20)           520
['dense_branch_depr3[0][0]']


 diabetes (Dense)               (None, 1)            21
['dense_branch_diab4[0][0]']


 depression (Dense)             (None, 1)            21
['dense_branch_depr4[0][0]']


==================================================================================
Total params: 17,277
Trainable params: 17,277
Non-trainable params: 0
__________________________________________________________________________________

tf.keras.utils.plot_model(model)
```

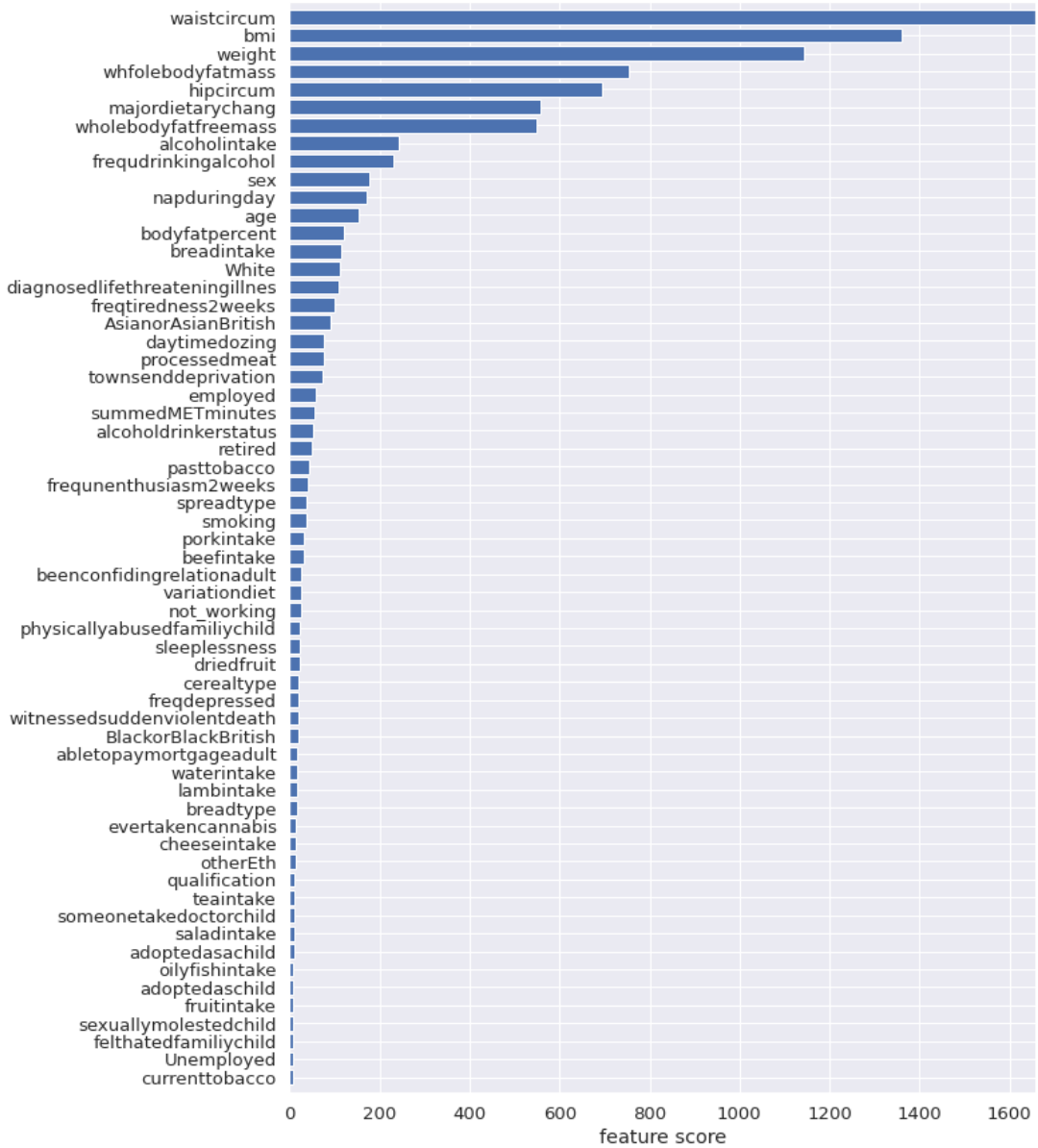

Dataset A
InputLayer
Dataset B
InputLayer
concatenate_1
Concatenate
dense_hidden_1
Dense
dropout_3
Dropout
dense_hidden_2
Dense
dropout_4
Dropout
dense_hidden_3
Dense
dropout_5
Dropout
dense_hidden_4
Dense

Experiment 3: Diabetes and CVDs

```
#Best feature selection

df = dfm.copy()
df['all'] =  df['diabetes'].astype(str) + df['cvd'].astype(str)
print("all unique: ", df['all'].value_counts())
#keep rows with only single and no disease subjects
df = df.loc[(df['all'] == '01') | (df['all'] == '10') | (df['all'] ==
'11')]
#reset the indexes of the newly created dataframe
df.reset_index(drop=True, inplace=True)


cols = df.columns.tolist()
cols.remove('diabetes')
cols.remove('depression')
cols.remove('cvd')
cols.remove('all')
df_X = df[cols]
df_Y = df[['all']]

# define feature selection
fs = SelectKBest(k=60).fit(df_X,df_Y)

X_selected = fs.transform(df_X)

# Get columns to keep and create new dataframe with those only
b_cols = fs.get_support(indices=True)
df_new = df_X.iloc[:,b_cols]
bestcols = df_new.columns.to_list()
scores = fs.scores_
li = df_X.columns.to_list()
scoresl = scores.tolist()
bestscore = []
for each in bestcols:
  bestscore.append(scoresl[li.index(each)])

res = {}
for each in bestcols:
  res[each] = bestscore[bestcols.index(each)]

ress = {k: v for k, v in sorted(res.items(), key=lambda item:
item[1])}

plt.figure(figsize=(10,15))
plt.margins(x=0, y=0.008)
plt.xlabel("feature score")
plt.barh(list(ress.keys()), list(ress.values()))
```

```
sn.set(font_scale=1.2)
plt.show()
```

```
all unique:  00    97144
01    53497
11     4455
10     1976
Name: all, dtype: int64
```




```
dfB_ = dfB.copy()
dfS_ = dfS.copy()
dfB_['all'] =  dfB_['diabetes'].astype(str) + dfB_['cvd'].astype(str)
dfS_['all'] =  dfS_['diabetes'].astype(str) + dfS_['cvd'].astype(str)
```

```
dfB_ = dfB_.loc[(dfB_['all'] == '10') | (dfB_['all'] == '01') |
(dfB_['all'] == '11')]
dfS_ = dfS_.loc[(dfS_['all'] == '10') | (dfS_['all'] == '01') |
(dfB_['all'] == '11')]


#keep the columns which are in bestcols
bigcols = []
for each in dfB_.columns.to_list():
  if each in bestcols:
    bigcols.append(each)

smallcols = []
for each in dfS_.columns.to_list():
  if each in bestcols:
    smallcols.append(each)

nB_cols = len(bigcols)
nS_cols = len(smallcols)

#reset the indexes of the newly created dataframes
dfB_.reset_index(drop=True, inplace=True)
dfS_.reset_index(drop=True, inplace=True)

print("shape of dfA_: ", dfB_.shape)
print("shape of dfB_: ", dfS_.shape)


print("-----Preparing initial datasets - Train and holdout-----")
cols = dfB_.columns.tolist()
cols.remove('diabetes')
cols.remove('depression')
cols.remove('cvd')

df_XB = dfB[bigcols]
df_YB = dfB[['diabetes', 'cvd']]

colsS = dfS.columns.tolist()
colsS.remove('diabetes')
colsS.remove('depression')
colsS.remove('cvd')

df_XS = dfS[smallcols]
df_YS = dfS[['diabetes', 'cvd']]


#prepare train and test dataset manually from the bigger dataset
test_size  = int(df_XB.shape[0] * 0.02)
```

```
df1 = df_YB.copy()
df1['out'] = df1['diabetes'].astype(str) + df1['cvd'].astype(str)
dis_cols  = df1['out'].unique().tolist()
rowsfortestB = []
for each in dis_cols:
  tempY = df1[df1.out == each]
  rows = np.random.choice(tempY.index.values,
int(test_size/len(dis_cols)))
  rowsfortestB.append(rows)

rowsfortestB = [j for i in rowsfortestB for j in i]
X_testB = df_XB.loc[rowsfortestB]
y_testB = df_YB.loc[rowsfortestB]
X_trainB = df_XB.loc[~df_XB.index.isin(rowsfortestB)]
y_trainB= df_YB.loc[~df_YB.index.isin(rowsfortestB)]

print("\nDistribution of Dataset B")
print("shape of X_trainB: ", X_trainB.shape)
print("shape of y_trainB: ", y_trainB.shape)
print("shape of X_testB: ", X_testB.shape)
print("shape of y_testB: ", y_testB.shape)

#prepare train and test dataset manually from the small dataset
df1S = df_YS.copy()
df1S['out'] = df1S['diabetes'].astype(str) +df1S['cvd'].astype(str)

rowsfortestS = []
for each in dis_cols:
  tempY = df1S[df1S.out == each]
  rows = np.random.choice(tempY.index.values,
int(test_size/len(dis_cols)))
  rowsfortestS.append(rows)

rowsfortestS = [j for i in rowsfortestS for j in i]
X_testS = df_XS.loc[rowsfortestS]
y_testS = df_YS.loc[rowsfortestS]
X_trainS = df_XS.loc[~df_XS.index.isin(rowsfortestS)]
y_trainS = df_YS.loc[~df_YS.index.isin(rowsfortestS)]

print("\nDistribution of Dataset S")
print("shape of X_trainS: ", X_trainS.shape)
print("shape of y_trainS: ", y_trainS.shape)
print("shape of X_testS: ", X_testS.shape)
print("shape of y_testS: ", y_testS.shape)


#get indices of categorical features

#Dataset S
```

```
cols = X_trainS.columns.tolist()
categorical_featS = []
for each in cols:
    if feat_cat[each] == 'cat':
        categorical_featS.append(cols.index(each))
print(categorical_featS)

#Datset B
cols = X_trainB.columns.tolist()
categorical_featB = []
for each in cols:
    if feat_cat[each] == 'cat':
        categorical_featB.append(cols.index(each))
print(categorical_featB)


#for smaller dataset
# perform a scaler transform on train and test dataset
trans = StandardScaler()
trans.fit(X_trainB)

data1 = trans.transform(X_trainB)
X_trainB = pd.DataFrame(data1)
#Apply same parameters of standardization on test as on train
data2 = trans.transform(X_testB)
X_testB = pd.DataFrame(data2)

#for smaller dataset
# perform a scaler transform on train and test dataset
trans1 = StandardScaler()
trans1.fit(X_trainS)

data1 = trans1.transform(X_trainS)
X_trainS = pd.DataFrame(data1)
#Apply same parameters of standardization on test as on train
data2 = trans1.transform(X_testS)
X_testS = pd.DataFrame(data2)


# Model configuration
batch_size = 512
no_epochs = 5
verbosity = 1
num_folds = 5

# Define per-fold score containers
acc_per_fold = []
loss_per_fold = []
```

```
# Define the K-fold Cross Validator
kfold = StratifiedKFold(n_splits=num_folds, shuffle=True)
rskf = RepeatedStratifiedKFold(n_splits=num_folds, n_repeats=2,
random_state=36851234)

# K-fold Cross Validation model evaluation
fold_no = 1

inputs = X_trainB.copy()
targets = y_trainB.copy()

inputsize = 0

# Define the layers of the model
input_A = Input(shape = (nB_cols,), name = 'Dataset A')
input_B = Input(shape = (nS_cols,), name = 'Dataset B')
out = Concatenate()([input_A, input_B])
dense_hidden_1 = Dense(55, activation = 'relu', name =
'dense_hidden_1')(out)
dense_hidden_1_d = Dropout(0.5)(dense_hidden_1)
dense_hidden_2 = Dense(50, activation = 'relu', name =
'dense_hidden_2')(dense_hidden_1_d)
dense_hidden_2_d = Dropout(0.25)(dense_hidden_2)
dense_hidden_3 = Dense(45, activation = 'relu', name =
'dense_hidden_3')(dense_hidden_2_d)
dense_hidden_3_d = Dropout(0.1)(dense_hidden_3)
dense_hidden_4 = Dense(40, activation = 'relu', name =
'dense_hidden_4')(dense_hidden_3_d)


dense_branch_diab1 = Dense(35, activation = 'relu', name =
'dense_branch_diab1')(dense_hidden_4)
#dense_hidden_diab1_d = Dropout(0.25)(dense_branch_diab1)
dense_branch_diab2 = Dense(30, activation = 'relu', name =
'dense_branch_diab2')(dense_branch_diab1)
diabetes = Dense(1, activation = 'sigmoid', name = 'diabetes')
(dense_branch_diab2)


dense_branch_cvd1 = Dense(35, activation = 'relu', name =
'dense_branch_cvd1')(dense_hidden_4)
#dense_hidden_cvd1_d = Dropout(0.25)(dense_branch_cvd1)
dense_branch_cvd2 = Dense(30, activation = 'relu', name =
'dense_branch_cvd2')(dense_branch_cvd1)
cvd = Dense(1, activation = 'sigmoid', name ='cvd')(dense_branch_cvd2)

for train, test in kfold.split(inputs, targets['diabetes']):
```

```
  trainX = inputs.iloc[train]
  trainy = targets.iloc[train]
  testXB = inputs.iloc[test]
  testyB = targets.iloc[test]
  #print("shape of trainX: ", trainX.shape)
  #print("trainy orig unique: \n", trainy.value_counts())

  # split the small dataset into train and validation
  X_train_s, X_tests, y_train_s, y_tests = train_test_split(X_trainS,
y_trainS, test_size=0.20,  stratify=y_trainS['diabetes'])


  #merge the target columns for undersampling
  df1 = trainy.copy()
  df1['out'] = df1['diabetes'].astype(str) +df1['cvd'].astype(str)

  #drop some 00 rows
  orig_count = df1['out'].value_counts().to_dict()
  orig_count['00'] = 0
  all_values = orig_count.values()
  orig_count['00'] = int(max(all_values))
  #orig_count['011'] = int(orig_count['011']*0.60)
  #print("orig_count after: ", orig_count)
  # define the undersampling method
  undersample = NearMiss(version=1, n_neighbors=3, sampling_strategy =
orig_count)
  trainy = df1['out']
  df_US_X, df_US_y = undersample.fit_resample(trainX, trainy)


  print("\nOversampling the dataset B using SMOTE-NC")
  #oversample the dataset
  samp_count = df1['out'].value_counts().to_dict()
  print("Class distribution before smote: ", samp_count)
  oversample = SMOTENC(categorical_features=categorical_featB)
  df_OS_X, df_OS_y = oversample.fit_resample(df_US_X, df_US_y)


  #split the 'out column to 3 target diseases
  a = pd.DataFrame(df_OS_y)
  a.columns = ['out']
  print("\nClass distribution in training - Dataset B:")
  print(a['out'].value_counts())
  #print("['out'] count after SMOTE: \n",a['out'].value_counts())
  a['diabetes'] = a['out'].str[:1]
  a['cvd'] = a['out'].str[1:2]


  #convert datatype of target columns to int
```

```
  a['diabetes'] = a['diabetes'].astype(str).astype(int)
  a['cvd'] = a['cvd'].astype(str).astype(int)
  a.drop(['out'], axis = 1)
  trainXB = df_OS_X.copy()
  trainyB = a.copy()



  #create input_S dataset for training
  print("\nPreparing training data for Dataset S")
  m = trainXB.shape[0]
  yb = trainyB.copy()
  yb['all'] = yb['diabetes'].astype(str) +yb['cvd'].astype(str)
  #print("y counts for train: \n", ys['all'].value_counts())
  yb_count = yb['all'].value_counts().to_dict()
  #print("yb_count: ", yb_count)
  S_y = y_train_s.copy()
  S_y['all'] = S_y['diabetes'].astype(str) +S_y['cvd'].astype(str)
  S_y_count = S_y['all'].value_counts().to_dict()
  #print("B_y['all'] counts: \n", B_y_count)

  colmov = S_y[['diabetes', 'cvd']]
  #print("shape of X_train_s: ", X_train_s.shape)
  #print("shape of S_y: ", S_y.shape)
  X_train_s.reset_index(drop=True, inplace=True)
  colmov.reset_index(drop=True, inplace=True)
  df_XSn = pd.concat([X_train_s, colmov], axis=1, join="inner")
  #print("shape of df_XSn: ", df_XSn.shape)
  #print("shape of B_y: ", B_y.shape)

  S_y_countNew = S_y_count.copy()
  for each in S_y_count:
    if S_y_count[each] > yb_count[each]:
      S_y_countNew[each] = yb_count[each]

  #some classes in S_y have more samples than in ys, drop the extra
samples
  undersample = NearMiss(version=1, n_neighbors=3, sampling_strategy =
S_y_countNew)
  df_S_X, df_S_y = undersample.fit_resample(df_XSn, S_y['all'])

  ros = RandomOverSampler(sampling_strategy = yb_count)

  X_res, y_res = ros.fit_resample(df_S_X, df_S_y)
  X_res = pd.DataFrame(X_res, columns = df_XSn.columns)
  y_res = pd.DataFrame(y_res)
  y_res.columns = ['all']

  #print("X_res shape: ", X_res.shape)
```

```
  #print("y_res shape: ", y_res.shape)
  y_res['newind'] = 0
  for each in yb_count:
    indexes = yb.loc[yb['all']==each].index.tolist()
    y_res.loc[y_res['all'] == each, 'newind'] = indexes
  X_res = pd.concat([X_res, y_res['newind']], axis=1, join="inner")
  X_res = X_res.sort_values('newind')
  X_res.reset_index(drop=True, inplace=True)
  X_res = X_res.drop(['diabetes', 'cvd','newind'], axis = 1)
  trainXS = X_res.copy()

  print("Dataset B train shape: ", trainXB.shape)
  print("Dataset S train shape: ", trainXS.shape)

  #create input_S dataset for validation
  print("\nPreparing training data for Dataset S")
  m = testXB.shape[0]
  yb = testyB.copy()
  yb['all'] = yb['diabetes'].astype(str) +yb['cvd'].astype(str)
  #print("y counts for train: \n", ys['all'].value_counts())
  yb_count = yb['all'].value_counts().to_dict()
  #print("yb_count: ", yb_count)
  S_y = y_tests.copy()
  S_y['all'] = S_y['diabetes'].astype(str) +S_y['cvd'].astype(str)
  S_y_count = S_y['all'].value_counts().to_dict()
  #print("B_y['all'] counts: \n", B_y_count)

  colmov = S_y[['diabetes', 'cvd']]
  #print("shape of X_train_s: ", X_train_s.shape)
  #print("shape of S_y: ", S_y.shape)
  X_tests.reset_index(drop=True, inplace=True)
  colmov.reset_index(drop=True, inplace=True)
  df_XSn = pd.concat([X_tests, colmov], axis=1, join="inner")
  #print("shape of df_XSn: ", df_XSn.shape)
  #print("shape of B_y: ", B_y.shape)

  S_y_countNew = S_y_count.copy()
  for each in S_y_count:
    if S_y_count[each] > yb_count[each]:
      S_y_countNew[each] = yb_count[each]

  #some classes in S_y have more samples than in ys, drop the extra
samples
  undersample = NearMiss(version=1, n_neighbors=3, sampling_strategy =
S_y_countNew)
  df_S_X, df_S_y = undersample.fit_resample(df_XSn, S_y['all'])

  ros = RandomOverSampler(sampling_strategy = yb_count)

  X_res, y_res = ros.fit_resample(df_S_X, df_S_y)
```

```
  X_res = pd.DataFrame(X_res, columns = df_XSn.columns)
  y_res = pd.DataFrame(y_res)
  y_res.columns = ['all']

  #print("X_res shape: ", X_res.shape)
  #print("y_res shape: ", y_res.shape)
  y_res['newind'] = 0
  for each in yb_count:
    indexes = yb.loc[yb['all']==each].index.tolist()
    y_res.loc[y_res['all'] == each, 'newind'] = indexes
  X_res = pd.concat([X_res, y_res['newind']], axis=1, join="inner")
  X_res = X_res.sort_values('newind')
  X_res.reset_index(drop=True, inplace=True)
  X_res = X_res.drop(['diabetes', 'cvd','newind'], axis = 1)
  testXS = X_res.copy()


  print("Dataset B validation shape: ", testXB.shape)
  print("Dataset S validation shape: ", testXS.shape)

  model = tf.keras.models.Model([input_A, input_B], [diabetes, cvd])

  initial_learning_rate = 0.1
  lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
      initial_learning_rate,
      decay_steps=10000,
      decay_rate=0.96,
      staircase=True)
  optimizersgd = tf.keras.optimizers.SGD(lr=0.0001, momentum=0.9)
  optimizeradam = tf.keras.optimizers.Adam(lr=0.00001)

  model.compile(
      loss={
          'diabetes': 'binary_crossentropy',
          'cvd': 'binary_crossentropy'
      },
      optimizer = optimizeradam,
      metrics = ['accuracy']
  )

  # Generate a print

print('------------------------------------------------------------------
---------')
  print(f'Training for fold {fold_no} ...')
  #print("len(df_OS_X): ", len(df_OS_X))
  #print("len(testX): ", len(testX))

  inputsize += trainXB.shape[0]
  # Fit data to model
```

```
  history = model.fit([trainXB, trainXS],
[trainyB['diabetes'],trainyB['cvd']],
            validation_data = ([testXB, testXS], [testyB['diabetes'],
testyB['cvd']]),
            batch_size=batch_size,
            epochs=no_epochs,
            verbose=0)

  # Generate generalization metrics
  scores = model.evaluate([testXB, testXS], [testyB['diabetes'],
testyB['cvd']], verbose=0)

#  print(f'Score for fold {fold_no}: {model.metrics_names[0]} of
{scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
# acc_per_fold.append(scores[1] * 100)
#  loss_per_fold.append(scores[0])

  # Increase fold number
  fold_no = fold_no + 1


#sort both test datasets to align target disease columns
y_testB.reset_index(drop=True, inplace=True)
X_testB.reset_index(drop=True, inplace=True)
y_testB['all'] = y_testB['diabetes'].astype(str)
+y_testB['cvd'].astype(str)

#create a dataframe for target variables of test B
all_testXB = pd.concat([X_testB, y_testB['all']], axis=1,
join="inner")
all_testXB.sort_values(by=['all'])
a = pd.DataFrame(columns=['all'])
a['all'] = all_testXB['all'].copy()
all_testXB = all_testXB.drop(['all'], axis = 1)
print("\nClass distribution in test - Dataset B:")
print(a['all'].value_counts())
#print("['out'] count after SMOTE: \n",a['out'].value_counts())
a['diabetes'] = a['all'].str[:1]
a['cvd'] = a['all'].str[1:2]
all_testyB = a.drop('all', axis=1)
#convert datatype of target columns to int
all_testyB['diabetes'] =
all_testyB['diabetes'].astype(str).astype(int)
all_testyB['cvd'] = all_testyB['cvd'].astype(str).astype(int)


y_testS.reset_index(drop=True, inplace=True)
X_testS.reset_index(drop=True, inplace=True)
y_testS['all'] = y_testS['diabetes'].astype(str)
```

```
+y_testS['cvd'].astype(str)
#create a dataframe for target variables of test S
all_testXS = pd.concat([X_testS, y_testS['all']], axis=1,
join="inner")
all_testXS.sort_values(by=['all'])
a = pd.DataFrame(columns=['all'])
a['all'] = all_testXS['all'].copy()
all_testXS = all_testXS.drop(['all'], axis = 1)
print("\nClass distribution in test - Dataset S:")
print(a['all'].value_counts())
#print("['out'] count after SMOTE: \n",a['out'].value_counts())
a['diabetes'] = a['all'].str[:1]
a['cvd'] = a['all'].str[1:2]
all_testyS = a.drop('all', axis=1)
#convert datatype of target columns to int
all_testyS['diabetes'] =
all_testyS['diabetes'].astype(str).astype(int)
all_testyS['cvd'] = all_testyS['cvd'].astype(str).astype(int)



preds = model.predict([all_testXB, all_testXS])
# Metrics for each target disease
target_v = ['diabetes', 'cvd']
print("input size: ", inputsize)

pred_threshold = {}

for each in target_v:
  print("\n-------------------------------------------------------")
  print(each)
  print("-------------------------------------------------------")

  #preds = model.predict(X_test)
  y_true = all_testyB[each]
  predictions = preds[target_v.index(each)]

  lr_probs = predictions

  fpr, tpr, thresholds = roc_curve(y_true, lr_probs)
  # calculate the g-mean for each threshold
  gmeans = np.sqrt(tpr * (1-fpr))
  # locate the index of the largest g-mean
  ix = np.argmax(gmeans)
  print('Best Threshold=%f, G-Mean=%.3f' % (thresholds[ix],
gmeans[ix]))
  # plot the roc curve for the model
  pyplot.plot([0,1], [0,1], linestyle='--', label='No Skill', zorder=-
1)
```

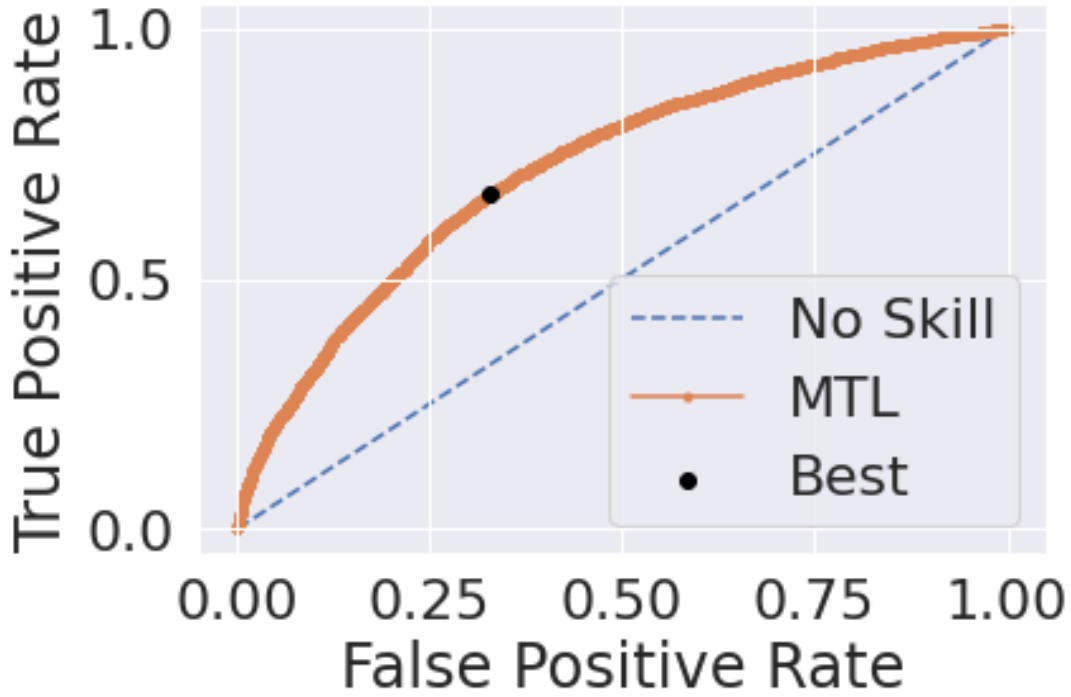
```
  pyplot.plot(fpr, tpr, marker='.', label='MTL',  zorder=-1)
  pyplot.scatter(fpr[ix], tpr[ix], marker='o', color='black',
label='Best', zorder=2)
  # axis labels
  pyplot.xlabel('False Positive Rate')
  pyplot.ylabel('True Positive Rate')
  pyplot.legend()
  # show the plot
  pyplot.show()

  #threshold based on ROC
  th = thresholds[ix]

  # calculate precision and recall for each threshold
  lr_precision, lr_recall, thresholds = precision_recall_curve(y_true,
lr_probs)
  # convert to f score
  fscore = (2 * lr_precision * lr_recall) / (lr_precision + lr_recall)
  ix = np.argmax(fscore)
  print('Best Threshold=%f, F-Score=%.3f' % (thresholds[ix],
fscore[ix]))
  # calculate scores
  #lr_f1, lr_auc = f1_score(y_over, y_pred), auc(lr_recall,
lr_precision)
  # summarize scores
  #print('NN: f1=%.3f auc=%.3f' % (lr_f1, lr_auc))
  # plot the roc curve for the model
  no_skill = len(y_true[y_true==1]) / len(y_true)
  pyplot.plot([0,1], [no_skill,no_skill], linestyle='--', label='No
Skill', zorder=-1)
  pyplot.plot(lr_recall, lr_precision, marker='.', label='MTL',
zorder=-1)
  pyplot.scatter(lr_recall[ix], lr_precision[ix], marker='o',
color='black', label='Best', zorder=2)
  # axis labels
  pyplot.xlabel('Recall')
  pyplot.ylabel('Precision')
  pyplot.legend()

  # show the plot
  pyplot.show()

  pred_threshold[each] = thresholds[ix]

  y_pred = [1 * (x[0]>th) for x in predictions]

  matrix = confusion_matrix(y_true, y_pred)
  print("y_test value counts:\n",Counter(y_true))
  print("y_pred value counts: ", Counter(y_pred))
  print("\nConfusion matrix: ")
```

```
  print(matrix)
  cmn = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]
  sn.heatmap(cmn, annot=True)
  sn.set(font_scale=2)
  plt.ylabel('Actual')
  plt.xlabel('Predicted')
  plt.show()

  tn, fp, fn, tp = matrix.ravel()
  recall = tp/(tp+fn)
  precision = tp/(fp+tp)
  specifcity = tn/(fp+tn)
  acc = (tn+tp)/(tn+fp+fn+tp)
  auc_score = auc(lr_recall, lr_precision)
  print("Precision: ", precision)
  print("Sensitivity/recall: ", recall)
  print("Specificity: ", specifcity)
  print("Accuracy: ", acc)
  print('PR AUC: %.3f' % auc_score)

  f_score = f1_score(y_true, y_pred)
  print("f_score: ", f_score)


shape of dfA_:  (223195, 71)
shape of dfB_:  (59976, 27)
-----Preparing initial datasets - Train and holdout-----

Distribution of Dataset B
shape of X_trainB:  (491789, 50)
shape of y_trainB:  (491789, 2)
shape of X_testB:  (10024, 50)
shape of y_testB:  (10024, 2)

Distribution of Dataset S
shape of X_trainS:  (149038, 10)
shape of y_trainS:  (149038, 2)
shape of X_testS:  (10024, 10)
shape of y_testS:  (10024, 2)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 3, 4, 5, 6, 7, 11, 12, 13, 14, 15, 16, 17, 19, 20, 23, 24, 25, 26,
27, 28, 29, 30, 31, 39, 41, 42, 43, 44, 45, 46, 47, 48, 49]

Oversampling the dataset B using SMOTE-NC
Class distribution before smote:  {'00': 220470, '01': 153338, '11':
15767, '10': 3856}

Class distribution in training - Dataset B:
00    153338
01    153338
```

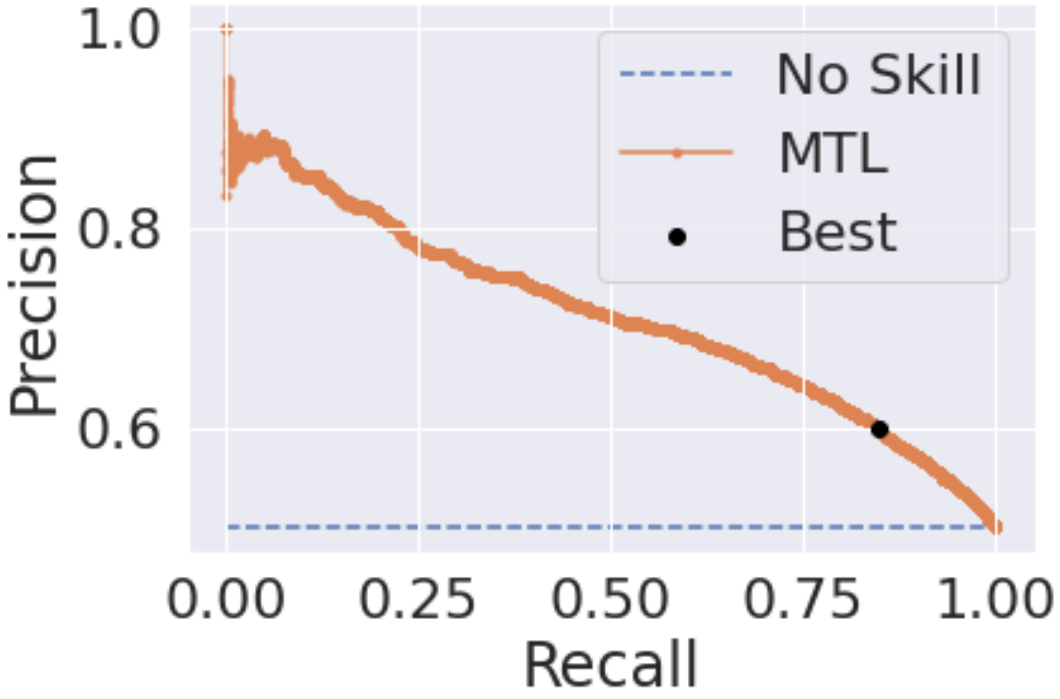
```
10    153338
11    153338
Name: out, dtype: int64

Preparing training data for Dataset S
Dataset B train shape:  (613352, 50)
Dataset S train shape:  (613352, 10)

Preparing training data for Dataset S
Dataset B validation shape:  (98358, 50)
Dataset S validation shape:  (98358, 10)
------------------------------------------------------------------------
--
Training for fold 1 ...

Oversampling the dataset B using SMOTE-NC
Class distribution before smote:  {'00': 220376, '01': 153432, '11':
15724, '10': 3899}

Class distribution in training - Dataset B:
00    153432
01    153432
10    153432
11    153432
Name: out, dtype: int64

Preparing training data for Dataset S
Dataset B train shape:  (613728, 50)
Dataset S train shape:  (613728, 10)

Preparing training data for Dataset S
Dataset B validation shape:  (98358, 50)
Dataset S validation shape:  (98358, 10)
------------------------------------------------------------------------
--
Training for fold 2 ...

Oversampling the dataset B using SMOTE-NC
Class distribution before smote:  {'00': 220240, '01': 153568, '11':
15695, '10': 3928}

Class distribution in training - Dataset B:
00    153568
01    153568
10    153568
11    153568
Name: out, dtype: int64

Preparing training data for Dataset S
```

```
Dataset B train shape:  (614272, 50)
Dataset S train shape:  (614272, 10)

Preparing training data for Dataset S
Dataset B validation shape:  (98358, 50)
Dataset S validation shape:  (98358, 10)
------------------------------------------------------------------------
--
Training for fold 3 ...

Oversampling the dataset B using SMOTE-NC
Class distribution before smote:  {'00': 220713, '01': 153095, '11':
15747, '10': 3876}

Class distribution in training - Dataset B:
00    153095
01    153095
10    153095
11    153095
Name: out, dtype: int64

Preparing training data for Dataset S
Dataset B train shape:  (612380, 50)
Dataset S train shape:  (612380, 10)

Preparing training data for Dataset S
Dataset B validation shape:  (98358, 50)
Dataset S validation shape:  (98358, 10)
------------------------------------------------------------------------
--
Training for fold 4 ...

Oversampling the dataset B using SMOTE-NC
Class distribution before smote:  {'00': 220389, '01': 153419, '11':
15743, '10': 3881}

Preparing training data for Dataset S
Dataset B train shape:  (613676, 50)
Dataset S train shape:  (613676, 10)

Preparing training data for Dataset S
Dataset B validation shape:  (98357, 50)
Dataset S validation shape:  (98357, 10)
------------------------------------------------------------------------
--
Training for fold 5 ...

Class distribution in test - Dataset B:
01    2506
```

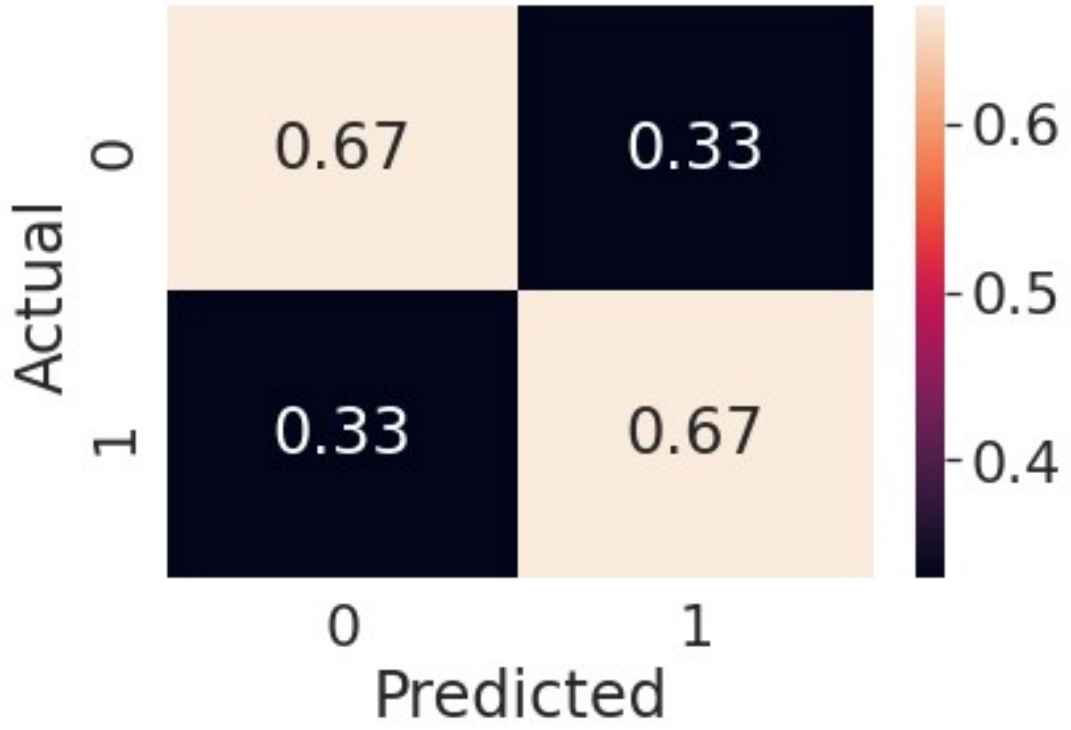
```
00    2506
11    2506
10    2506
Name: all, dtype: int64

Class distribution in test - Dataset S:
01    2506
00    2506
11    2506
10    2506
Name: all, dtype: int64
input size:  3067408

----------------------------------------------------------
diabetes
----------------------------------------------------------
Best Threshold=0.331352, G-Mean=0.671
```



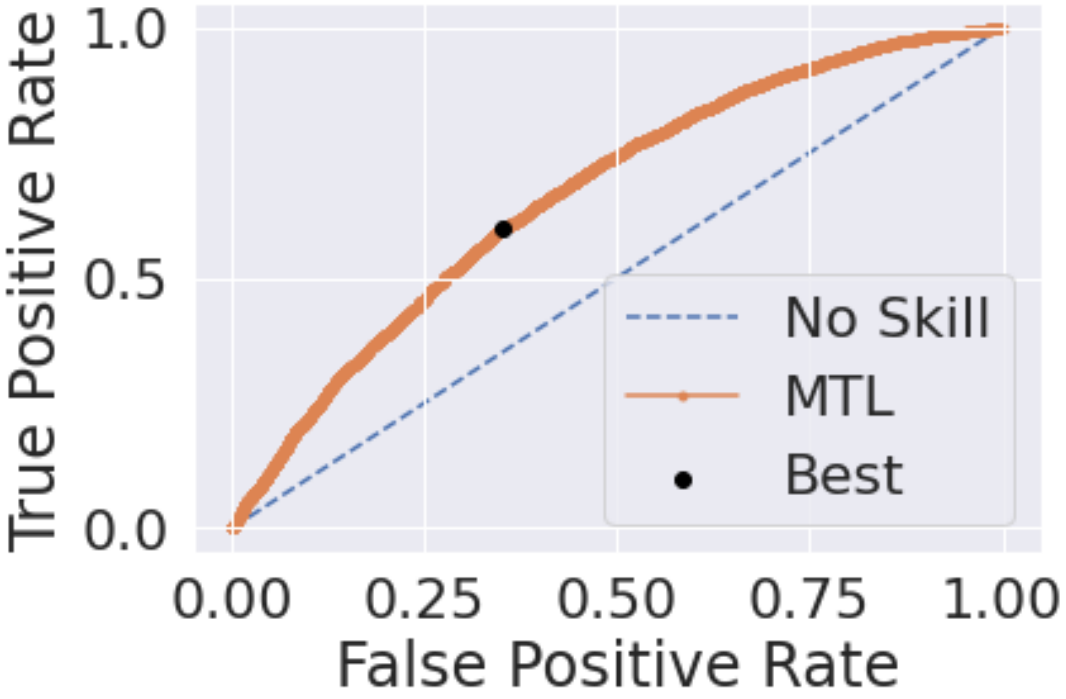

```
Best Threshold=0.196659, F-Score=0.703
```



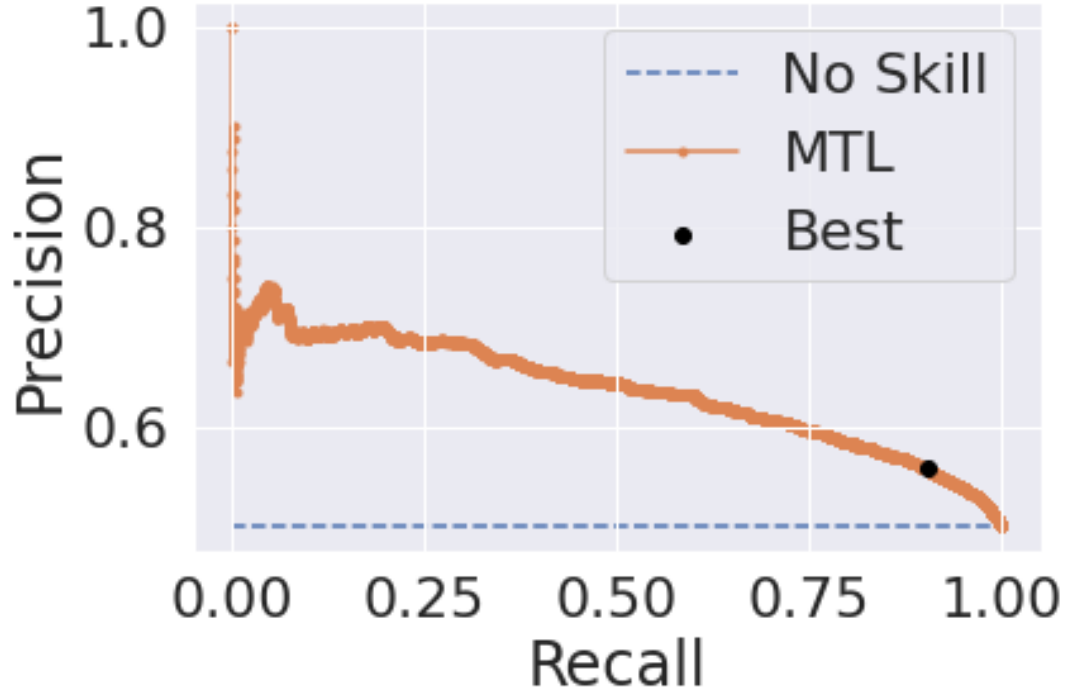

```
y_test value counts:
 Counter({0: 5012, 1: 5012})
y_pred value counts:  Counter({0: 5013, 1: 5011})

Confusion matrix:
[[3362 1650]
 [1651 3361]]
```




```
Precision:  0.6707244063061265
Sensitivity/recall:  0.6705905826017557
Specificity:  0.6707901037509976
Accuracy:  0.6706903431763767
PR AUC: 0.713
f_score:  0.6706574877781103

------------------------------------------------------
cvd
------------------------------------------------------
Best Threshold=0.545159, G-Mean=0.625
```

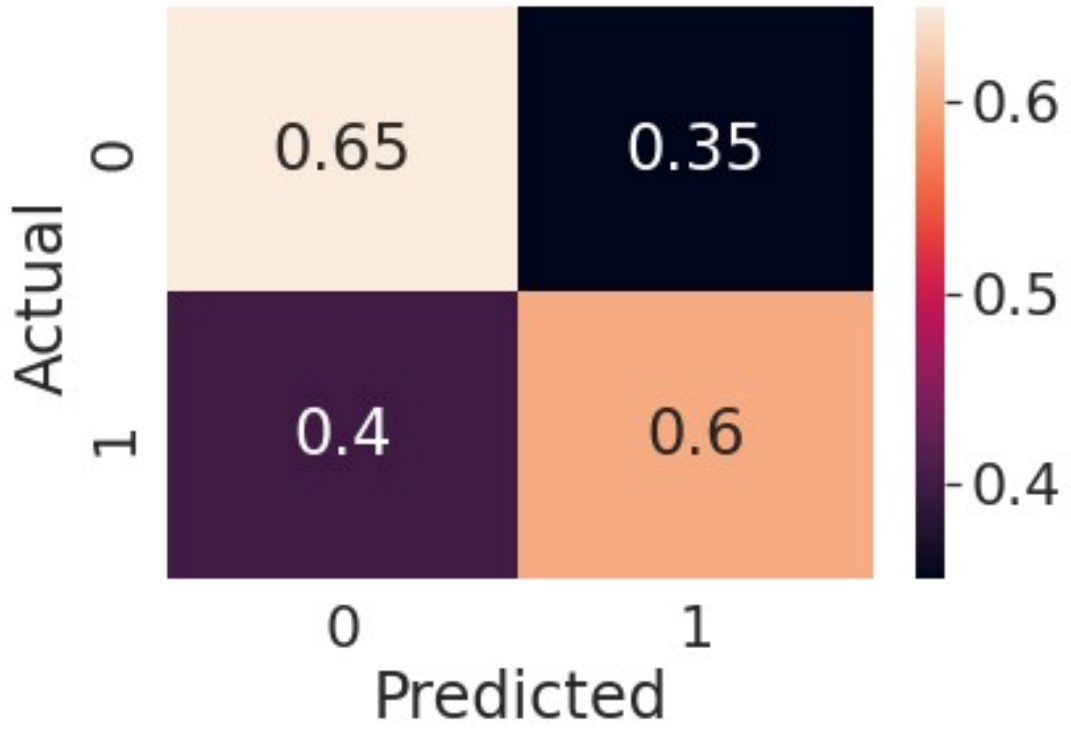



```
Best Threshold=0.317774, F-Score=0.690
```




```
y_test value counts:
 Counter({1: 5012, 0: 5012})
y_pred value counts:  Counter({0: 5262, 1: 4762})
```

```
Confusion matrix:
[[3258 1754]
 [2004 3008]]
```



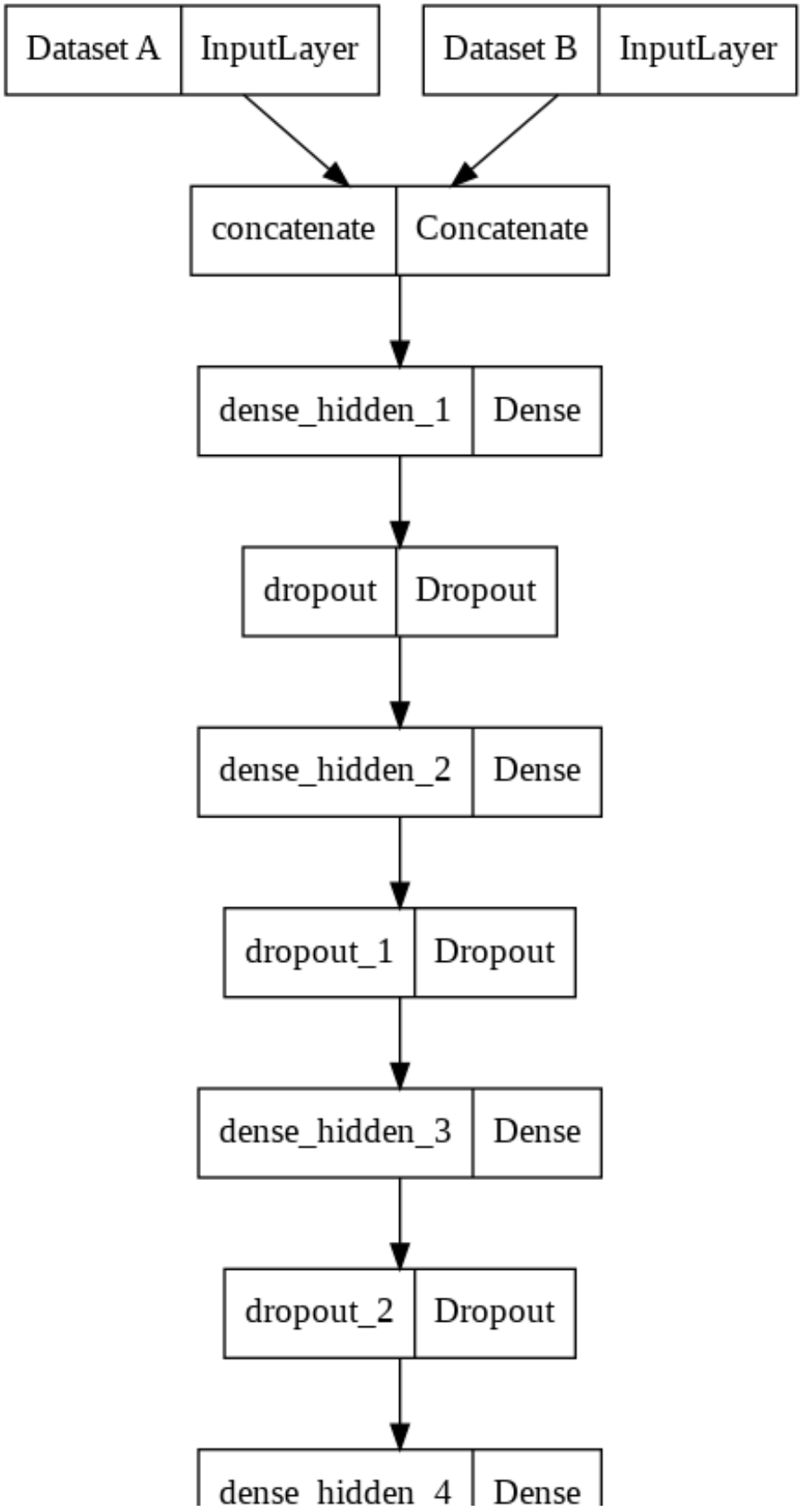

```
Precision:  0.631667366652667
Sensitivity/recall:  0.6001596169193935
Specificity:  0.6500399042298484
Accuracy:  0.6250997605746209
PR AUC: 0.637
f_score:  0.6155105381624718
```

```
print("Number of layers: ", len(model.layers))
model.summary()
```

```
Number of layers:  16
Model: "model_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
Connected to
==================================================================
 Dataset A (InputLayer)      [(None, 50)]              0           []
```

```
 Dataset B (InputLayer)         [(None, 10)]         0           []



 concatenate_2 (Concatenate)    (None, 60)           0
['Dataset A[0][0]',

'Dataset B[0][0]']


 dense_hidden_1 (Dense)         (None, 55)           3355
['concatenate_2[0][0]']


 dropout_6 (Dropout)            (None, 55)           0
['dense_hidden_1[0][0]']


 dense_hidden_2 (Dense)         (None, 50)           2800
['dropout_6[0][0]']


 dropout_7 (Dropout)            (None, 50)           0
['dense_hidden_2[0][0]']


 dense_hidden_3 (Dense)         (None, 45)           2295
['dropout_7[0][0]']


 dropout_8 (Dropout)            (None, 45)           0
['dense_hidden_3[0][0]']


 dense_hidden_4 (Dense)         (None, 40)           1840
['dropout_8[0][0]']


 dense_branch_diab1 (Dense)     (None, 35)           1435
['dense_hidden_4[0][0]']


 dense_branch_cvd1 (Dense)      (None, 35)           1435
['dense_hidden_4[0][0]']


 dense_branch_diab2 (Dense)     (None, 30)           1080
['dense_branch_diab1[0][0]']
```

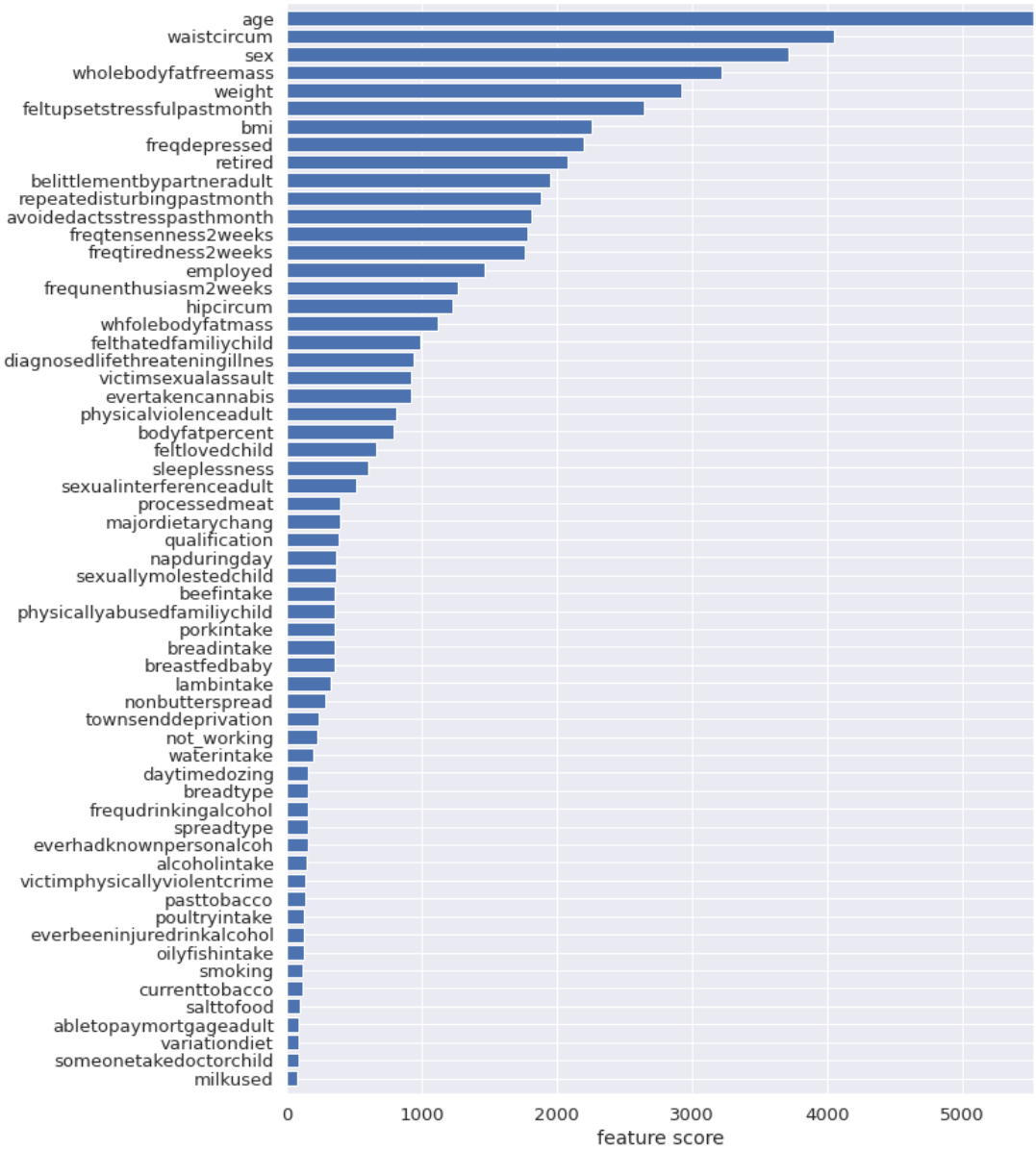
```
 dense_branch_cvd2 (Dense)      (None, 30)           1080
['dense_branch_cvd1[0][0]']


 diabetes (Dense)               (None, 1)            31
['dense_branch_diab2[0][0]']


 cvd (Dense)                    (None, 1)            31
['dense_branch_cvd2[0][0]']


==================================================================================
============================
Total params: 15,382
Trainable params: 15,382
Non-trainable params: 0
__________________________________________________________________________________
_____________________________

tf.keras.utils.plot_model(model)
```


Dataset A
InputLayer
Dataset B
InputLayer
concatenate
Concatenate
dense_hidden_1
Dense
dropout
Dropout
dense_hidden_2
Dense
dropout_1
Dropout
dense_hidden_3
Dense
dropout_2
Dropout
dense_hidden_4
Dense

Experiment 4: Depression and CVD

```
#Best feature selection
df = dfm.copy()
df['all'] =  df['depression'].astype(str) +df['cvd'].astype(str)
print("all unique: ", df['all'].value_counts())
#keep rows with only single and no disease subjects
df = df.loc[(df['all'] == '10') | (df['all'] == '11') | (df['all'] ==
'01')]
#reset the indexes of the newly created dataframe
df.reset_index(drop=True, inplace=True)
print("all unique: ", df['all'].value_counts())


cols = df.columns.tolist()
cols.remove('diabetes')
cols.remove('depression')
cols.remove('cvd')
cols.remove('all')
df_X = df[cols]
df_Y = df[['all']]

# define feature selection
fs = SelectKBest(k=60).fit(df_X,df_Y)

X_selected = fs.transform(df_X)

# Get columns to keep and create new dataframe with those only
b_cols = fs.get_support(indices=True)
df_new = df_X.iloc[:,b_cols]
bestcols = df_new.columns.to_list()
scores = fs.scores_
li = df_X.columns.to_list()
scoresl = scores.tolist()
bestscore = []
for each in bestcols:
  bestscore.append(scoresl[li.index(each)])

res = {}
for each in bestcols:
  res[each] = bestscore[bestcols.index(each)]
ress = {k: v for k, v in sorted(res.items(), key=lambda item:
item[1])}

plt.figure(figsize=(10,15))
plt.margins(x=0, y=0.008)
plt.xlabel("feature score")
plt.barh(list(ress.keys()), list(ress.values()))
sn.set(font_scale=1.2)
plt.show()
```

```
all unique:  00    52728
10    46392
01    29536
11    28416
Name: all, dtype: int64
all unique:  10    46392
01    29536
11    28416
Name: all, dtype: int64
```




```
dfB_ = dfB.copy()
dfS_ = dfS.copy()
dfB_['all'] =  dfB_['depression'].astype(str) +
```

```
dfB_['cvd'].astype(str)
dfS_['all'] =  dfS_['depression'].astype(str) +
dfS_['cvd'].astype(str)

dfB_ = dfB_.loc[(dfB_['all'] == '10') | (dfB_['all'] == '01') |
(dfB_['all'] == '11')]
dfS_ = dfS_.loc[(dfS_['all'] == '10') | (dfS_['all'] == '01') |
(dfB_['all'] == '11')]


#keep the columns which are in bestcols
bigcols = []
for each in dfB_.columns.to_list():
  if each in bestcols:
    bigcols.append(each)

smallcols = []
for each in dfS_.columns.to_list():
  if each in bestcols:
    smallcols.append(each)

nB_cols = len(bigcols)
nS_cols = len(smallcols)

#reset the indexes of the newly created dataframes
dfB_.reset_index(drop=True, inplace=True)
dfS_.reset_index(drop=True, inplace=True)

print("shape of dfA_: ", dfB_.shape)
print("shape of dfB_: ", dfS_.shape)


print("-----Preparing initial datasets - Train and holdout-----")
cols = dfB_.columns.tolist()
cols.remove('diabetes')
cols.remove('depression')
cols.remove('cvd')

df_XB = dfB[bigcols]
df_YB = dfB[['depression', 'cvd']]

colsS = dfS.columns.tolist()
colsS.remove('diabetes')
colsS.remove('depression')
colsS.remove('cvd')

df_XS = dfS[smallcols]
df_YS = dfS[['depression', 'cvd']]
```

```
#prepare train and test dataset manually from the bigger dataset
test_size  = int(df_XB.shape[0] * 0.02)
df1 = df_YB.copy()
df1['out'] = df1['depression'].astype(str) + df1['cvd'].astype(str)
dis_cols  = df1['out'].unique().tolist()
rowsfortestB = []
for each in dis_cols:
  tempY = df1[df1.out == each]
  rows = np.random.choice(tempY.index.values,
int(test_size/len(dis_cols)))
  rowsfortestB.append(rows)

rowsfortestB = [j for i in rowsfortestB for j in i]
X_testB = df_XB.loc[rowsfortestB]
y_testB = df_YB.loc[rowsfortestB]
X_trainB = df_XB.loc[~df_XB.index.isin(rowsfortestB)]
y_trainB= df_YB.loc[~df_YB.index.isin(rowsfortestB)]

print("\nDistribution of Dataset B")
print("shape of X_trainB: ", X_trainB.shape)
print("shape of y_trainB: ", y_trainB.shape)
print("shape of X_testB: ", X_testB.shape)
print("shape of y_testB: ", y_testB.shape)

#prepare train and test dataset manually from the small dataset
df1S = df_YS.copy()
df1S['out'] = df1S['depression'].astype(str) +df1S['cvd'].astype(str)

rowsfortestS = []
for each in dis_cols:
  tempY = df1S[df1S.out == each]
  rows = np.random.choice(tempY.index.values,
int(test_size/len(dis_cols)))
  rowsfortestS.append(rows)

rowsfortestS = [j for i in rowsfortestS for j in i]
X_testS = df_XS.loc[rowsfortestS]
y_testS = df_YS.loc[rowsfortestS]
X_trainS = df_XS.loc[~df_XS.index.isin(rowsfortestS)]
y_trainS = df_YS.loc[~df_YS.index.isin(rowsfortestS)]

print("\nDistribution of Dataset S")
print("shape of X_trainS: ", X_trainS.shape)
print("shape of y_trainS: ", y_trainS.shape)
print("shape of X_testS: ", X_testS.shape)
print("shape of y_testS: ", y_testS.shape)
```

```
#get indices of categorical features

#Dataset S
cols = X_trainS.columns.tolist()
categorical_featS = []
for each in cols:
    if feat_cat[each] == 'cat':
        categorical_featS.append(cols.index(each))

#Datset B
cols = X_trainB.columns.tolist()
categorical_featB = []
for each in cols:
    if feat_cat[each] == 'cat':
        categorical_featB.append(cols.index(each))


#for smaller dataset
# perform a scaler transform on train and test dataset
trans = StandardScaler()
trans.fit(X_trainB)

data1 = trans.transform(X_trainB)
X_trainB = pd.DataFrame(data1)
#Apply same parameters of standardization on test as on train
data2 = trans.transform(X_testB)
X_testB = pd.DataFrame(data2)

#for smaller dataset
# perform a scaler transform on train and test dataset
trans1 = StandardScaler()
trans1.fit(X_trainS)

data1 = trans1.transform(X_trainS)
X_trainS = pd.DataFrame(data1)
#Apply same parameters of standardization on test as on train
data2 = trans1.transform(X_testS)
X_testS = pd.DataFrame(data2)


# Model configuration
batch_size = 500
no_epochs = 5
verbosity = 1
num_folds = 5

# Define per-fold score containers
acc_per_fold = []
loss_per_fold = []
```

```
# Define the K-fold Cross Validator
kfold = StratifiedKFold(n_splits=num_folds, shuffle=True)
rskf = RepeatedStratifiedKFold(n_splits=num_folds, n_repeats=2,
random_state=36851234)

# K-fold Cross Validation model evaluation
fold_no = 1

inputs = X_trainB.copy()
targets = y_trainB.copy()

inputsize = 0

# Define the layers of the model
input_A = Input(shape = (nB_cols,), name = 'Dataset A')
input_B = Input(shape = (nS_cols,), name = 'Dataset B')
out = Concatenate()([input_A, input_B])
dense_hidden_1 = Dense(55, activation = 'relu', name =
'dense_hidden_1')(out)
dense_hidden_1_d = Dropout(0.5)(dense_hidden_1)
dense_hidden_2 = Dense(50, activation = 'relu', name =
'dense_hidden_2')(dense_hidden_1_d)
dense_hidden_2_d = Dropout(0.25)(dense_hidden_2)
dense_hidden_3 = Dense(45, activation = 'relu', name =
'dense_hidden_3')(dense_hidden_2_d)
dense_hidden_3_d = Dropout(0.1)(dense_hidden_3)
dense_hidden_4 = Dense(40, activation = 'relu', name =
'dense_hidden_4')(dense_hidden_3_d)


dense_branch_diab1 = Dense(35, activation = 'relu', name =
'dense_branch_diab1')(dense_hidden_4)
#dense_hidden_diab1_d = Dropout(0.25)(dense_branch_diab1)
dense_branch_diab2 = Dense(30, activation = 'relu', name =
'dense_branch_diab2')(dense_branch_diab1)
dense_branch_diab3 = Dense(25, activation = 'relu', name =
'dense_branch_diab3')(dense_branch_diab2)
dense_branch_diab4 = Dense(20, activation = 'relu', name =
'dense_branch_diab4')(dense_branch_diab2)
diabetes = Dense(1, activation = 'sigmoid', name = 'diabetes')
(dense_branch_diab4)


dense_branch_depr1 = Dense(35, activation = 'relu', name =
'dense_branch_depr1')(dense_hidden_4)
#dense_hidden_depr1_d = Dropout(0.1)(dense_branch_depr1)
dense_branch_depr2 = Dense(30, activation = 'relu', name =
```

```
'dense_branch_depr2')(dense_branch_depr1)
dense_branch_depr3 = Dense(25, activation = 'relu', name =
'dense_branch_depr3')(dense_branch_depr2)
dense_branch_depr4 = Dense(20, activation = 'relu', name =
'dense_branch_depr4')(dense_branch_depr3)
depression = Dense(1, activation = 'sigmoid', name ='depression')
(dense_branch_depr4)

dense_branch_cvd1 = Dense(35, activation = 'relu', name =
'dense_branch_cvd1')(dense_hidden_4)
#dense_hidden_cvd1_d = Dropout(0.25)(dense_branch_cvd1)
dense_branch_cvd2 = Dense(30, activation = 'relu', name =
'dense_branch_cvd2')(dense_branch_cvd1)
dense_branch_cvd3 = Dense(25, activation = 'relu', name =
'dense_branch_cvd3')(dense_branch_cvd2)
dense_branch_cvd4 = Dense(20, activation = 'relu', name =
'dense_branch_cvd4')(dense_branch_cvd3)
cvd = Dense(1, activation = 'sigmoid', name ='cvd')(dense_branch_cvd4)

for train, test in kfold.split(inputs, targets['cvd']):

  trainX = inputs.iloc[train]
  trainy = targets.iloc[train]
  testXB = inputs.iloc[test]
  testyB = targets.iloc[test]
  #print("shape of trainX: ", trainX.shape)
  #print("trainy orig unique: \n", trainy.value_counts())

  # split the small dataset into train and validation
  X_train_s, X_tests, y_train_s, y_tests = train_test_split(X_trainS,
y_trainS, test_size=0.20,  stratify=y_trainS['cvd'])


  #merge the target columns for undersampling
  df1 = trainy.copy()
  df1['out'] = df1['depression'].astype(str) +df1['cvd'].astype(str)

  #drop some 00 rows
  orig_count = df1['out'].value_counts().to_dict()
  orig_count['00'] = 0
  all_values = orig_count.values()
  orig_count['00'] = int(max(all_values))

  # define the undersampling method
  undersample = NearMiss(version=1, n_neighbors=3, sampling_strategy =
orig_count)
  trainy = df1['out']
  df_US_X, df_US_y = undersample.fit_resample(trainX, trainy)
```

```
print("\nOversampling the dataset B using SMOTE-NC")
#oversample the dataset
samp_count = df1['out'].value_counts().to_dict()
print("Class distribution before smote: ", samp_count)

oversample = SMOTENC(categorical_features=categorical_featB)
df_OS_X, df_OS_y = oversample.fit_resample(df_US_X, df_US_y)


#split the 'out column to 3 target diseases
a = pd.DataFrame(df_OS_y)
a.columns = ['out']
print("\nClass distribution in training - Dataset B:")
#print("['out'] count after SMOTE: \n",a['out'].value_counts())
a['depression'] = a['out'].str[:1]
a['cvd'] = a['out'].str[1:2]


#convert datatype of target columns to int
a['depression'] = a['depression'].astype(str).astype(int)
a['cvd'] = a['cvd'].astype(str).astype(int)

trainXB = df_OS_X.copy()
trainyB = a.copy()

#create input_S dataset for training
print("\nPreparing training data for Dataset S")
m = trainXB.shape[0]
yb = trainyB.copy()
yb['all'] = yb['depression'].astype(str) +yb['cvd'].astype(str)
#print("y counts for train: \n", ys['all'].value_counts())
yb_count = yb['all'].value_counts().to_dict()
#print("yb_count: ", yb_count)
S_y = y_train_s.copy()
S_y['all'] = S_y['depression'].astype(str) +S_y['cvd'].astype(str)
S_y_count = S_y['all'].value_counts().to_dict()
#print("B_y['all'] counts: \n", B_y_count)

colmov = S_y[['depression', 'cvd']]
X_train_s.reset_index(drop=True, inplace=True)
colmov.reset_index(drop=True, inplace=True)
df_XSn = pd.concat([X_train_s, colmov], axis=1, join="inner")
#print("shape of df_XSn: ", df_XSn.shape)
#print("shape of B_y: ", B_y.shape)

S_y_countNew = S_y_count.copy()
for each in S_y_count:
  if S_y_count[each] > yb_count[each]:
    S_y_countNew[each] = yb_count[each]
```

```
  #some classes in S_y have more samples than in ys, drop the extra
samples
  undersample = NearMiss(version=1, n_neighbors=3, sampling_strategy =
S_y_countNew)
  df_S_X, df_S_y = undersample.fit_resample(df_XSn, S_y['all'])

  ros = RandomOverSampler(sampling_strategy = yb_count)

  X_res, y_res = ros.fit_resample(df_S_X, df_S_y)
  X_res = pd.DataFrame(X_res, columns = df_XSn.columns)
  y_res = pd.DataFrame(y_res)
  y_res.columns = ['all']

  #print("X_res shape: ", X_res.shape)
  #print("y_res shape: ", y_res.shape)
  y_res['newind'] = 0
  for each in yb_count:
    indexes = yb.loc[yb['all']==each].index.tolist()
    y_res.loc[y_res['all'] == each, 'newind'] = indexes
  X_res = pd.concat([X_res, y_res['newind']], axis=1, join="inner")
  X_res = X_res.sort_values('newind')
  X_res.reset_index(drop=True, inplace=True)
  X_res = X_res.drop(['depression', 'cvd','newind'], axis = 1)
  trainXS = X_res.copy()

  print("Dataset B train shape: ", trainXB.shape)
  print("Dataset S train shape: ", trainXS.shape)

  #create input_S dataset for validation
  print("\nPreparing training data for Dataset S")
  m = testXB.shape[0]
  yb = testyB.copy()
  yb['all'] = yb['depression'].astype(str) +yb['cvd'].astype(str)
  #print("y counts for train: \n", ys['all'].value_counts())
  yb_count = yb['all'].value_counts().to_dict()
  #print("yb_count: ", yb_count)
  S_y = y_tests.copy()
  S_y['all'] = S_y['depression'].astype(str) +S_y['cvd'].astype(str)
  S_y_count = S_y['all'].value_counts().to_dict()
  #print("B_y['all'] counts: \n", B_y_count)

  colmov = S_y[['depression', 'cvd']]
  #print("shape of X_train_s: ", X_train_s.shape)
  #print("shape of S_y: ", S_y.shape)
  X_tests.reset_index(drop=True, inplace=True)
  colmov.reset_index(drop=True, inplace=True)
  df_XSn = pd.concat([X_tests, colmov], axis=1, join="inner")
  #print("shape of df_XSn: ", df_XSn.shape)
  #print("shape of B_y: ", B_y.shape)
```

```
  S_y_countNew = S_y_count.copy()
  for each in S_y_count:
    if S_y_count[each] > yb_count[each]:
      S_y_countNew[each] = yb_count[each]

  #some classes in S_y have more samples than in ys, drop the extra
samples
  undersample = NearMiss(version=1, n_neighbors=3, sampling_strategy =
S_y_countNew)
  df_S_X, df_S_y = undersample.fit_resample(df_XSn, S_y['all'])

  ros = RandomOverSampler(sampling_strategy = yb_count)

  X_res, y_res = ros.fit_resample(df_S_X, df_S_y)
  X_res = pd.DataFrame(X_res, columns = df_XSn.columns)
  y_res = pd.DataFrame(y_res)
  y_res.columns = ['all']

  #print("X_res shape: ", X_res.shape)
  #print("y_res shape: ", y_res.shape)
  y_res['newind'] = 0
  for each in yb_count:
    indexes = yb.loc[yb['all']==each].index.tolist()
    y_res.loc[y_res['all'] == each, 'newind'] = indexes
  X_res = pd.concat([X_res, y_res['newind']], axis=1, join="inner")
  X_res = X_res.sort_values('newind')
  X_res.reset_index(drop=True, inplace=True)
  X_res = X_res.drop(['depression', 'cvd','newind'], axis = 1)
  testXS = X_res.copy()


  print("Dataset B validation shape: ", testXB.shape)
  print("Dataset S validation shape: ", testXS.shape)

  model = tf.keras.models.Model([input_A, input_B], [depression, cvd])

  initial_learning_rate = 0.1
  lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
      initial_learning_rate,
      decay_steps=10000,
      decay_rate=0.96,
      staircase=True)
  optimizersgd = tf.keras.optimizers.SGD(lr=0.0001, momentum=0.9)
  optimizeradam = tf.keras.optimizers.Adam(lr=0.0001)

  model.compile(
      loss={
          'depression': 'binary_crossentropy',
```

```
            'cvd': 'binary_crossentropy'
        },
        optimizer = optimizeradam,
        metrics = ['accuracy']
    )

    # Generate a print

print('------------------------------------------------------------------------
---------')
    print(f'Training for fold {fold_no} ...')
    #print("len(df_OS_X): ", len(df_OS_X))
    #print("len(testX): ", len(testX))

    inputsize += trainXB.shape[0]
    # Fit data to model
    history = model.fit([trainXB, trainXS],
[trainyB['depression'],trainyB['cvd']],
              validation_data = ([testXB, testXS],
[testyB['depression'], testyB['cvd']]),
              batch_size=batch_size,
              epochs=no_epochs,
              verbose=0)

    # Generate generalization metrics
    scores = model.evaluate([testXB, testXS], [testyB['depression'],
testyB['cvd']], verbose=0)


    # Increase fold number
    fold_no = fold_no + 1



#sort both test datasets to align target disease columns
y_testB.reset_index(drop=True, inplace=True)
X_testB.reset_index(drop=True, inplace=True)
y_testB['all'] = y_testB['depression'].astype(str)
+y_testB['cvd'].astype(str)

#create a dataframe for target variables of test B
all_testXB = pd.concat([X_testB, y_testB['all']], axis=1,
join="inner")
all_testXB.sort_values(by=['all'])
a = pd.DataFrame(columns=['all'])
a['all'] = all_testXB['all'].copy()
all_testXB = all_testXB.drop(['all'], axis = 1)
print("\nClass distribution in test - Dataset B:")
print(a['all'].value_counts())
```

```
#print("['out'] count after SMOTE: \n",a['out'].value_counts())
a['depression'] = a['all'].str[:1]
a['cvd'] = a['all'].str[1:2]
all_testyB = a.drop('all', axis=1)
#convert datatype of target columns to int
all_testyB['depression'] =
all_testyB['depression'].astype(str).astype(int)
all_testyB['cvd'] = all_testyB['cvd'].astype(str).astype(int)


y_testS.reset_index(drop=True, inplace=True)
X_testS.reset_index(drop=True, inplace=True)
y_testS['all'] = y_testS['depression'].astype(str)
+y_testS['cvd'].astype(str)
#create a dataframe for target variables of test S
all_testXS = pd.concat([X_testS, y_testS['all']], axis=1,
join="inner")
all_testXS.sort_values(by=['all'])
a = pd.DataFrame(columns=['all'])
a['all'] = all_testXS['all'].copy()
all_testXS = all_testXS.drop(['all'], axis = 1)
print("\nClass distribution in test - Dataset S:")
print(a['all'].value_counts())
#print("['out'] count after SMOTE: \n",a['out'].value_counts())
a['depression'] = a['all'].str[:1]
a['cvd'] = a['all'].str[1:2]
all_testyS = a.drop('all', axis=1)
#convert datatype of target columns to int
all_testyS['depression'] =
all_testyS['depression'].astype(str).astype(int)
all_testyS['cvd'] = all_testyS['cvd'].astype(str).astype(int)



preds = model.predict([all_testXB, all_testXS])
# Metrics for each target disease
target_v = ['depression', 'cvd']

pred_threshold = {}
print("input size: ", inputsize)

for each in target_v:
  print("\n-------------------------------------------------------")
  print(each)
  print("-------------------------------------------------------")

  #preds = model.predict(X_test)
  y_true = all_testyB[each]
  predictions = preds[target_v.index(each)]
```

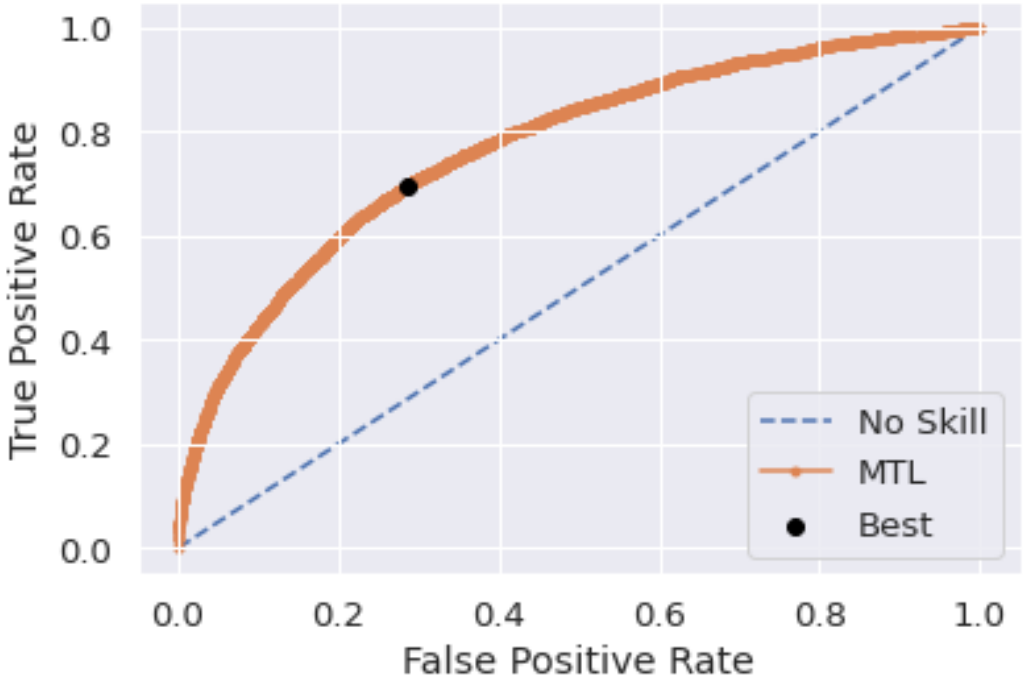
```
  lr_probs = predictions

  fpr, tpr, thresholds = roc_curve(y_true, lr_probs)
  # calculate the g-mean for each threshold
  gmeans = np.sqrt(tpr * (1-fpr))
  # locate the index of the largest g-mean
  ix = np.argmax(gmeans)
  print('Best Threshold=%f, G-Mean=%.3f' % (thresholds[ix],
gmeans[ix]))
  # plot the roc curve for the model
  pyplot.plot([0,1], [0,1], linestyle='--', label='No Skill', zorder=-
1)
  pyplot.plot(fpr, tpr, marker='.', label='MTL',  zorder=-1)
  pyplot.scatter(fpr[ix], tpr[ix], marker='o', color='black',
label='Best', zorder=2)
  # axis labels
  pyplot.xlabel('False Positive Rate')
  pyplot.ylabel('True Positive Rate')
  pyplot.legend()
  # show the plot
  pyplot.show()

  # calculate precision and recall for each threshold
  lr_precision, lr_recall, thresholds = precision_recall_curve(y_true,
lr_probs)
  # convert to f score
  fscore = (2 * lr_precision * lr_recall) / (lr_precision + lr_recall)
  ix = np.argmax(fscore)
  print('Best Threshold=%f, F-Score=%.3f' % (thresholds[ix],
fscore[ix]))
  # calculate scores
  #lr_f1, lr_auc = f1_score(y_over, y_pred), auc(lr_recall,
lr_precision)
  # summarize scores
  #print('NN: f1=%.3f auc=%.3f' % (lr_f1, lr_auc))
  # plot the roc curve for the model
  no_skill = len(y_true[y_true==1]) / len(y_true)
  pyplot.plot([0,1], [no_skill,no_skill], linestyle='--', label='No
Skill', zorder=-1)
  pyplot.plot(lr_recall, lr_precision, marker='.', label='MTL',
zorder=-1)
  pyplot.scatter(lr_recall[ix], lr_precision[ix], marker='o',
color='black', label='Best', zorder=2)
  # axis labels
  pyplot.xlabel('Recall')
  pyplot.ylabel('Precision')
  pyplot.legend()
  # show the plot
  pyplot.show()
```

```
  pred_threshold[each] = thresholds[ix]

  y_pred = [1 * (x[0]>thresholds[ix]) for x in predictions]

  matrix = confusion_matrix(y_true, y_pred)
  print("y_test value counts:\n",Counter(y_true))
  print("y_pred value counts: ", Counter(y_pred))
  print("\nConfusion matrix: ")
  print(matrix)
  cmn = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]
  sn.heatmap(cmn, annot=True)
  sn.set(font_scale=2)
  plt.ylabel('Actual')
  plt.xlabel('Predicted')
  plt.show()

  tn, fp, fn, tp = matrix.ravel()
  recall = tp/(tp+fn)
  precision = tp/(fp+tp)
  specifcity = tn/(fp+tn)
  acc = (tn+tp)/(tn+fp+fn+tp)
  auc_score = auc(lr_recall, lr_precision)
  print("Precision: ", precision)
  print("Sensitivity/recall: ", recall)
  print("Specificity: ", specifcity)
  print("Accuracy: ", acc)
  print('PR AUC: %.3f' % auc_score)

  f_score = f1_score(y_true, y_pred)
  print("f_score: ", f_score)


shape of dfA_:  (324932, 71)
shape of dfB_:  (90296, 27)
-----Preparing initial datasets - Train and holdout-----

Distribution of Dataset B
shape of X_trainB:  (491338, 41)
shape of y_trainB:  (491338, 2)
shape of X_testB:  (10024, 41)
shape of y_testB:  (10024, 2)

Distribution of Dataset S
shape of X_trainS:  (147579, 19)
shape of y_trainS:  (147579, 2)
shape of X_testS:  (10024, 19)
shape of y_testS:  (10024, 2)

Oversampling the dataset B using SMOTE-NC
```

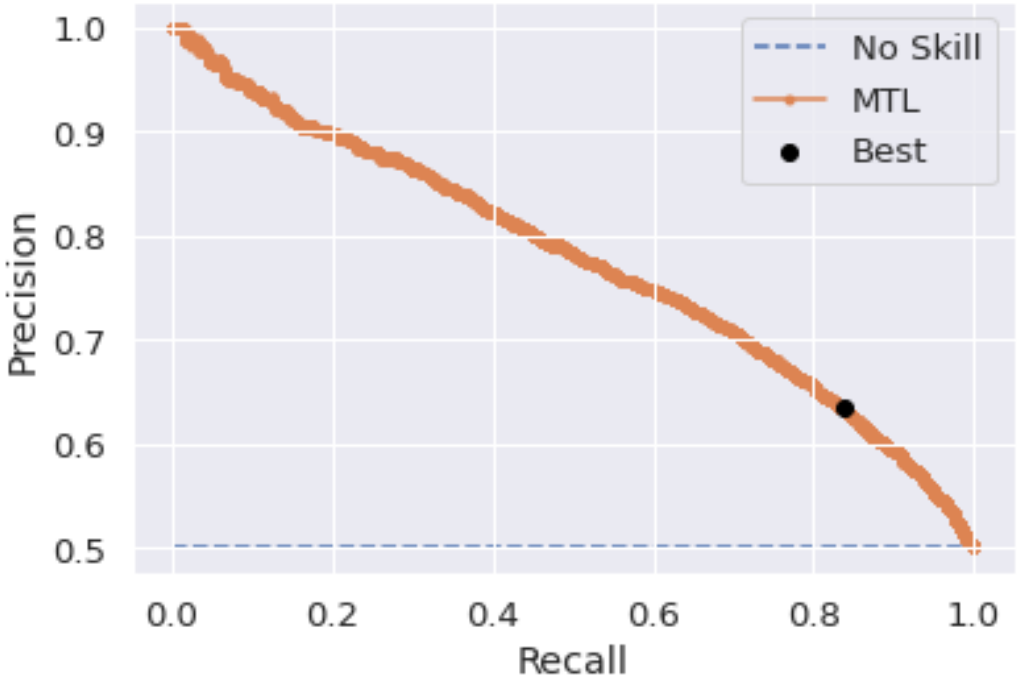
```
Class distribution before smote:  {'00': 139267, '01': 100029, '10':
84775, '11': 68999}

Class distribution in training - Dataset B:

Preparing training data for Dataset S
Dataset B train shape:  (400116, 41)
Dataset S train shape:  (400116, 19)

Preparing training data for Dataset S
Dataset B validation shape:  (98268, 41)
Dataset S validation shape:  (98268, 19)
------------------------------------------------------------------------
--
Training for fold 1 ...

Oversampling the dataset B using SMOTE-NC
Class distribution before smote:  {'00': 139089, '01': 99920, '10':
84953, '11': 69108}

Class distribution in training - Dataset B:

Preparing training data for Dataset S
Dataset B train shape:  (399680, 41)
Dataset S train shape:  (399680, 19)

Preparing training data for Dataset S
Dataset B validation shape:  (98268, 41)
Dataset S validation shape:  (98268, 19)
------------------------------------------------------------------------
--
Training for fold 2 ...

Oversampling the dataset B using SMOTE-NC
Class distribution before smote:  {'00': 139135, '01': 99973, '10':
84907, '11': 69055}

Class distribution in training - Dataset B:

Preparing training data for Dataset S
Dataset B train shape:  (399892, 41)
Dataset S train shape:  (399892, 19)

Preparing training data for Dataset S
Dataset B validation shape:  (98268, 41)
Dataset S validation shape:  (98268, 19)
------------------------------------------------------------------------
--
Training for fold 3 ...
```

```
Oversampling the dataset B using SMOTE-NC
Class distribution before smote:  {'00': 138692, '01': 99982, '10':
85351, '11': 69046}

Class distribution in training - Dataset B:

Preparing training data for Dataset S
Dataset B train shape:  (399928, 41)
Dataset S train shape:  (399928, 19)

Preparing training data for Dataset S
Dataset B validation shape:  (98267, 41)
Dataset S validation shape:  (98267, 19)
------------------------------------------------------------------------
--
Training for fold 4 ...

Oversampling the dataset B using SMOTE-NC
Class distribution before smote:  {'00': 139093, '01': 99944, '10':
84950, '11': 69084}

Class distribution in training - Dataset B:

Preparing training data for Dataset S
Dataset B train shape:  (399776, 41)
Dataset S train shape:  (399776, 19)

Preparing training data for Dataset S
Dataset B validation shape:  (98267, 41)
Dataset S validation shape:  (98267, 19)
------------------------------------------------------------------------
--
Training for fold 5 ...

Class distribution in test - Dataset B:
11    2506
01    2506
00    2506
10    2506
Name: all, dtype: int64

Class distribution in test - Dataset S:
11    2506
01    2506
00    2506
10    2506
Name: all, dtype: int64
input size:  1999392
```

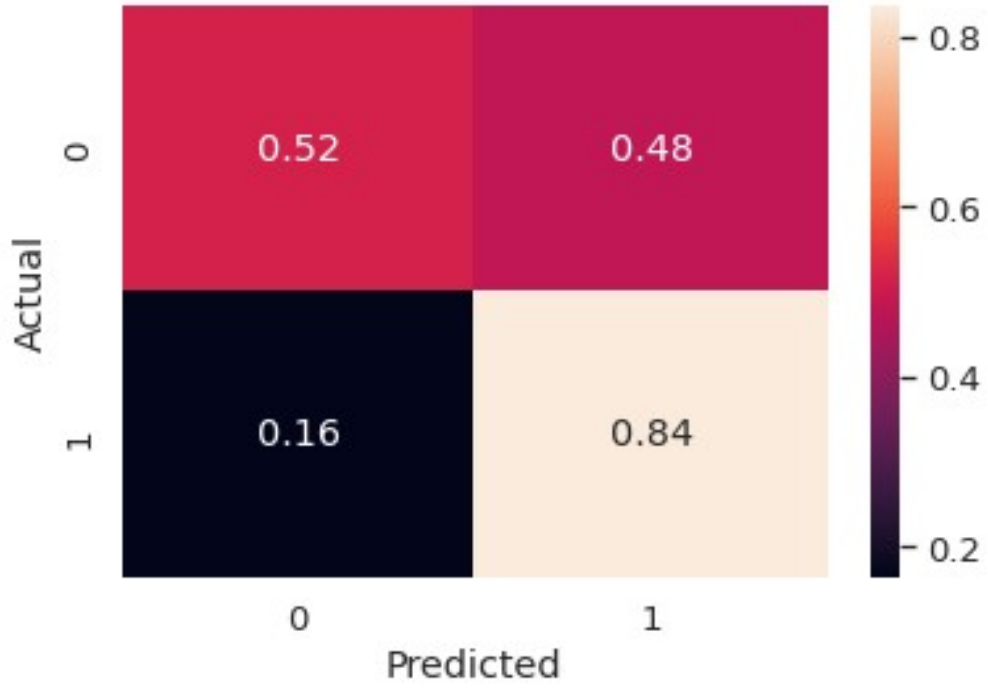
```
------------------------------------------------------
depression
------------------------------------------------------
Best Threshold=0.480855, G-Mean=0.706
```



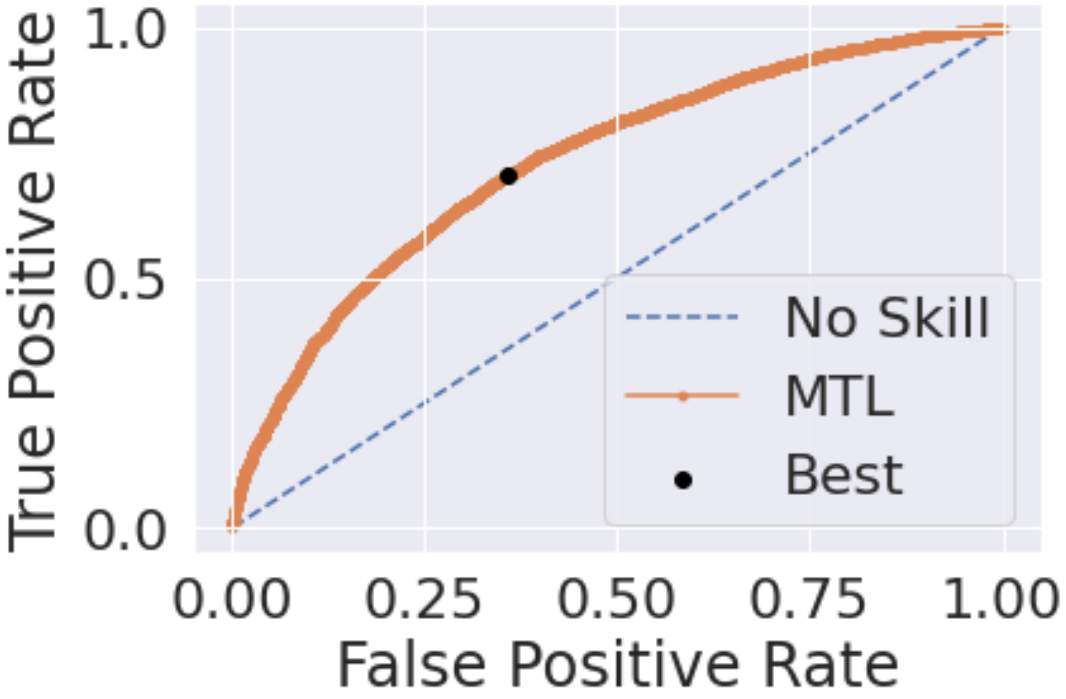

```
Best Threshold=0.353970, F-Score=0.722
```



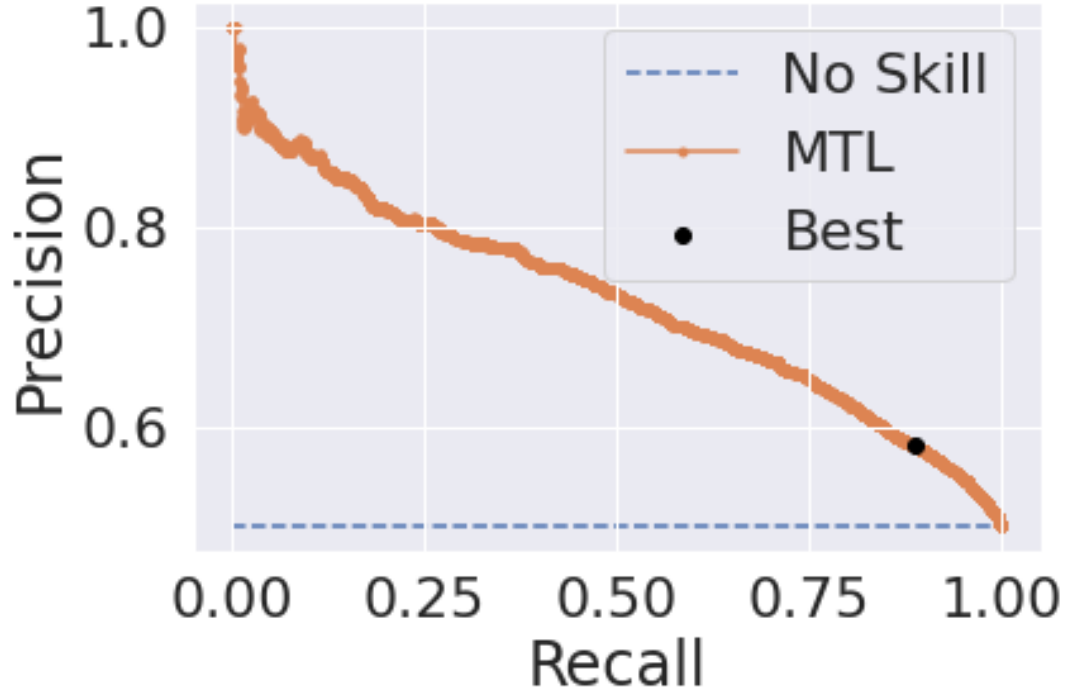

```
y_test value counts:
 Counter({1: 5012, 0: 5012})
y_pred value counts:  Counter({1: 6621, 0: 3403})

Confusion matrix:
[[2589 2423]
 [ 814 4198]]
```




```
Precision:  0.6340431958918592
Sensitivity/recall:  0.8375897845171588
Specificity:  0.5165602553870711
Accuracy:  0.6770750199521149
PR AUC: 0.776
f_score:  0.7217398779334651

-------------------------------------------------------
cvd
-------------------------------------------------------
Best Threshold=0.453173, G-Mean=0.675
```

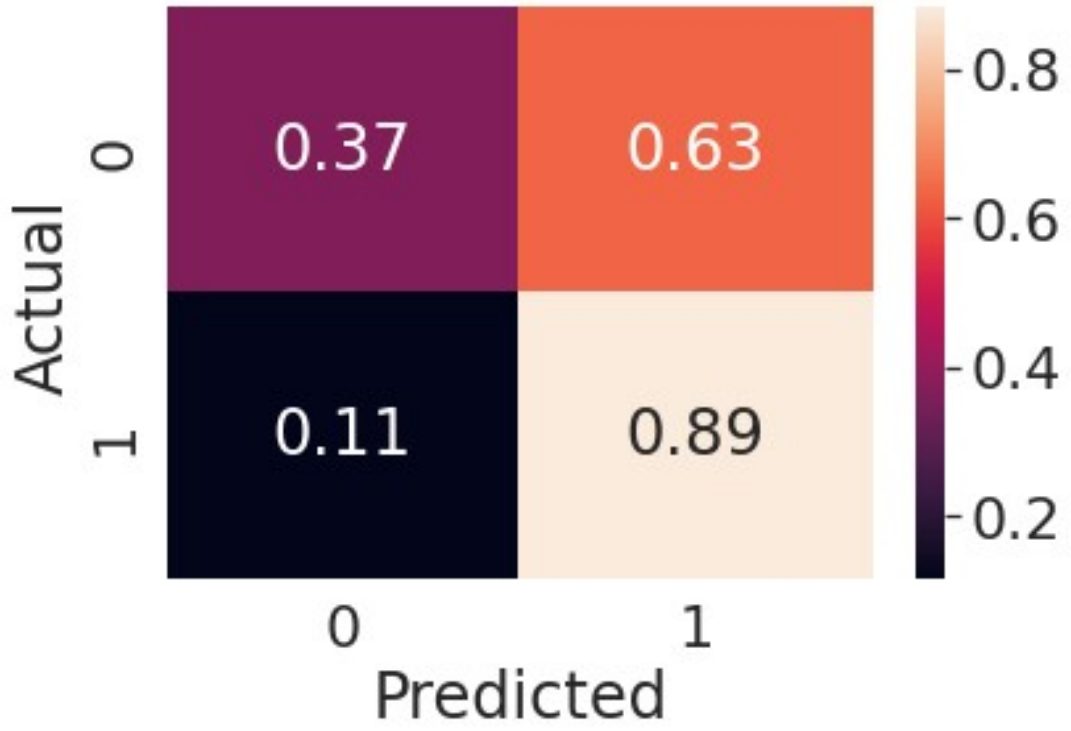



```
Best Threshold=0.330206, F-Score=0.703
```




```
y_test value counts:
 Counter({1: 5012, 0: 5012})
y_pred value counts:  Counter({1: 7617, 0: 2407})
```

```
Confusion matrix:
[[1833 3179]
 [ 574 4438]]
```




```
Precision:  0.5826440855980045
Sensitivity/recall:  0.8854748603351955
Specificity:  0.36572226656025536
Accuracy:  0.6255985634477255
PR AUC: 0.726
f_score:  0.7028268271438752
```

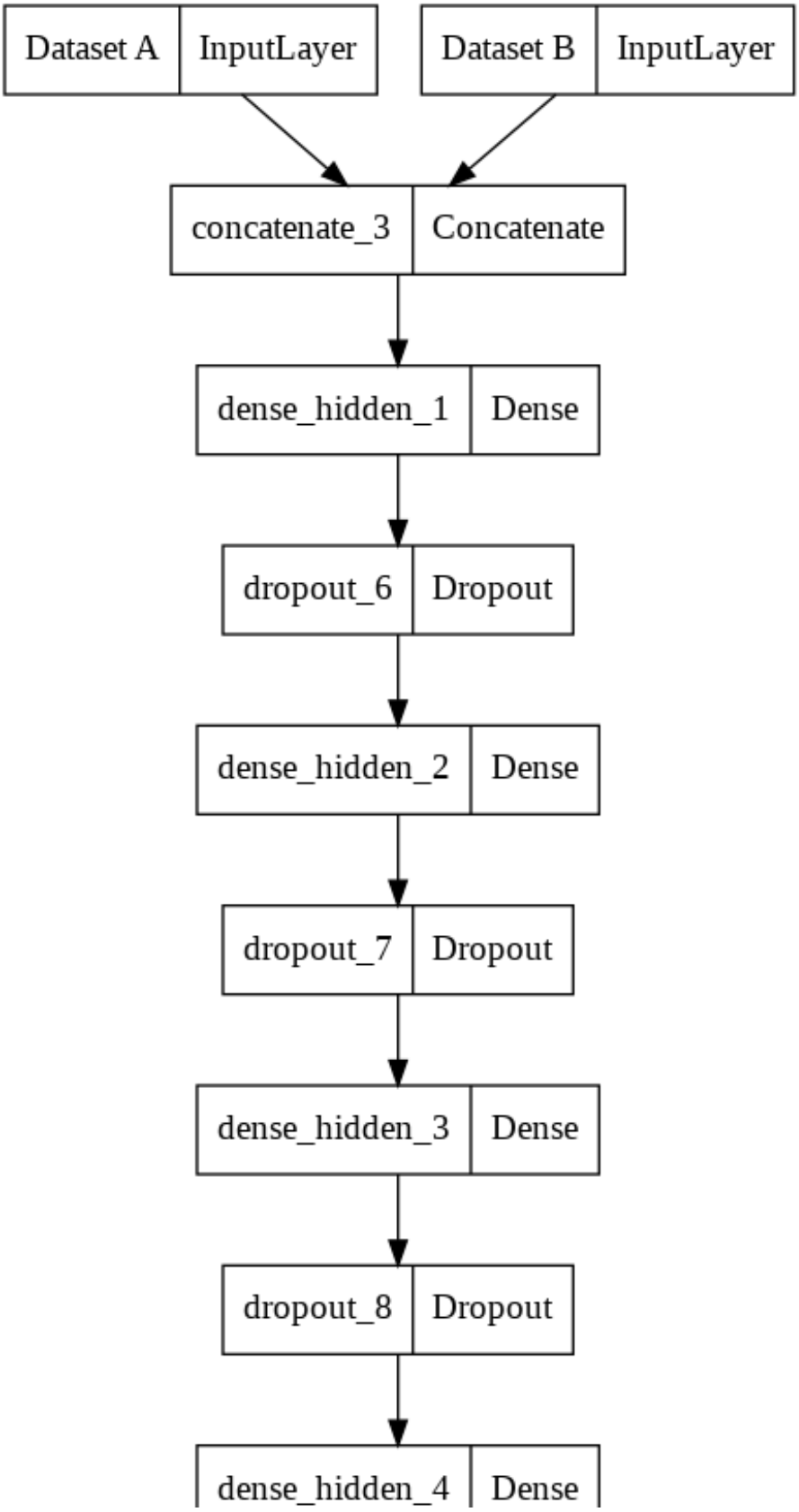
```
print("Number of layers: ", len(model.layers))
model.summary()
```

```
Number of layers:  20
Model: "model_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
Connected to
==================================================================
 Dataset A (InputLayer)      [(None, 41)]              0           []
```

```
 Dataset B (InputLayer)         [(None, 19)]         0           []



 concatenate_3 (Concatenate)    (None, 60)           0
['Dataset A[0][0]',

'Dataset B[0][0]']


 dense_hidden_1 (Dense)         (None, 55)           3355
['concatenate_3[0][0]']


 dropout_9 (Dropout)            (None, 55)           0
['dense_hidden_1[0][0]']


 dense_hidden_2 (Dense)         (None, 50)           2800
['dropout_9[0][0]']


 dropout_10 (Dropout)           (None, 50)           0
['dense_hidden_2[0][0]']


 dense_hidden_3 (Dense)         (None, 45)           2295
['dropout_10[0][0]']


 dropout_11 (Dropout)           (None, 45)           0
['dense_hidden_3[0][0]']


 dense_hidden_4 (Dense)         (None, 40)           1840
['dropout_11[0][0]']


 dense_branch_depr1 (Dense)     (None, 35)           1435
['dense_hidden_4[0][0]']


 dense_branch_cvd1 (Dense)      (None, 35)           1435
['dense_hidden_4[0][0]']


 dense_branch_depr2 (Dense)     (None, 30)           1080
['dense_branch_depr1[0][0]']
```

```
 dense_branch_cvd2 (Dense)      (None, 30)           1080
['dense_branch_cvd1[0][0]']


 dense_branch_depr3 (Dense)     (None, 25)           775
['dense_branch_depr2[0][0]']


 dense_branch_cvd3 (Dense)      (None, 25)           775
['dense_branch_cvd2[0][0]']


 dense_branch_depr4 (Dense)     (None, 20)           520
['dense_branch_depr3[0][0]']


 dense_branch_cvd4 (Dense)      (None, 20)           520
['dense_branch_cvd3[0][0]']


 depression (Dense)             (None, 1)            21
['dense_branch_depr4[0][0]']


 cvd (Dense)                    (None, 1)            21
['dense_branch_cvd4[0][0]']


==================================================================================
Total params: 17,952
Trainable params: 17,952
Non-trainable params: 0
__________________________________________________________________________________
```

```
tf.keras.utils.plot_model(model)
```


Dataset A
InputLayer
Dataset B
InputLayer
concatenate_3
Concatenate
dense_hidden_1
Dense
dropout_6
Dropout
dense_hidden_2
Dense
dropout_7
Dropout
dense_hidden_3
Dense
dropout_8
Dropout
dense_hidden_4
Dense

Bibliography

[1] Emerging Risk Factors Collaboration et al. «Diabetes mellitus, fasting blood glucose concentration, and risk of vascular disease: a collaborative meta-analysis of 102 prospective studies». In: *The Lancet* 375.9733 (2010), pp. 2215–2222 (cit. on p. 1).

[2] Gitanjali M Singh et al. «The age-specific quantitative effects of metabolic risk factors on cardiovascular diseases and diabetes: a pooled analysis». In: *PloS one* 8.7 (2013), e65174 (cit. on p. 1).

[3] *GBD results tool*. URL: `http://ghdx.healthdata.org/gbd-results-tool?params=gbd-api-2019-permalink%2Fd780dffbe8a381b25e1416884959e88b` (cit. on p. 1).

[4] Dominique L Musselman, Dwight L Evans, and Charles B Nemeroff. «The relationship of depression to cardiovascular disease: epidemiology, biology, and treatment». In: *Archives of general psychiatry* 55.7 (1998), pp. 580–592 (cit. on p. 1).

[5] Alexander H Glassman. «Depression and cardiovascular comorbidity». In: *Dialogues in clinical neuroscience* 9.1 (2007), p. 9 (cit. on p. 1).

[6] Wayne J Katon. «The comorbidity of diabetes mellitus and depression». In: *The American journal of medicine* 121.11 (2008), S8–S15 (cit. on p. 1).

[7] Sherita Hill Golden, Mariana Lazo, Mercedes Carnethon, Alain G Bertoni, Pamela J Schreiner, Ana V Diez Roux, Hochang Benjamin Lee, and Constantine Lyketsos. «Examining a bidirectional association between depressive symptoms and diabetes». In: *Jama* 299.23 (2008), pp. 2751–2759 (cit. on p. 2).

[8] Anne Engum. «The role of depression and anxiety in onset of diabetes in a large population-based study». In: *Journal of psychosomatic research* 62.1 (2007), pp. 31–38 (cit. on p. 2).

[9] Briana Mezuk, William W Eaton, Sandra Albrecht, and Sherita Hill Golden. «Depression and type 2 diabetes over the lifespan: a meta-analysis». In: *Diabetes care* 31.12 (2008), pp. 2383–2390 (cit. on p. 2).

[10] P De Jonge, JF Roy, P Saz, G Marcos, and A Lobo. «Prevalent and incident depression in community-dwelling elderly persons with diabetes mellitus: results from the ZARADEMP project». In: *Diabetologia* 49.11 (2006), pp. 2627–2633 (cit. on p. 2).

[11] Mirjam J Knol, Eibert R Heerdink, Antoine CG Egberts, Mirjam I Geerlings, Kees J Gorter, Mattijs E Numans, Diederick E Grobbee, Olaf H Klungel, and Huibert Burger. «Depressive symptoms in subjects with diagnosed and undiagnosed type 2 diabetes». In: *Psychosomatic medicine* 69.4 (2007), pp. 300–305 (cit. on p. 2).

[12] Arie Nouwen, Kirsty Winkley, J Twisk, CE Lloyd, Mark Peyrot, K Ismail, and F Pouwer. «Type 2 diabetes mellitus as a risk factor for the onset of depression: a systematic review and meta-analysis». In: *Diabetologia* 53.12 (2010), pp. 2480–2486 (cit. on p. 2).

[13] Xiang Wang, Fei Wang, and Jianying Hu. «A multi-task learning framework for joint disease risk prediction and comorbidity discovery». In: *2014 22nd International Conference on Pattern Recognition*. IEEE. 2014, pp. 220–225 (cit. on p. 4).

[14] Ilayda Beyreli, Oguzhan Karakahya, and A Ercument Cicek. «Deep multitask learning of gene risk for comorbid neurodevelopmental disorders». In: *bioRxiv* (2021), pp. 2020–06 (cit. on p. 4).

[15] Adrian Benton, Margaret Mitchell, and Dirk Hovy. «Multi-task learning for mental health using social media text». In: *arXiv preprint arXiv:1712.03538* (2017) (cit. on p. 4).

[16] *International Classification of Diseases (ICD)*. URL: `https://www.who.int/classifications/classification-of-diseases` (cit. on p. 8).