CSE439-Introduction to Deep Learning

Salih Can AYDOĞDU

191805061

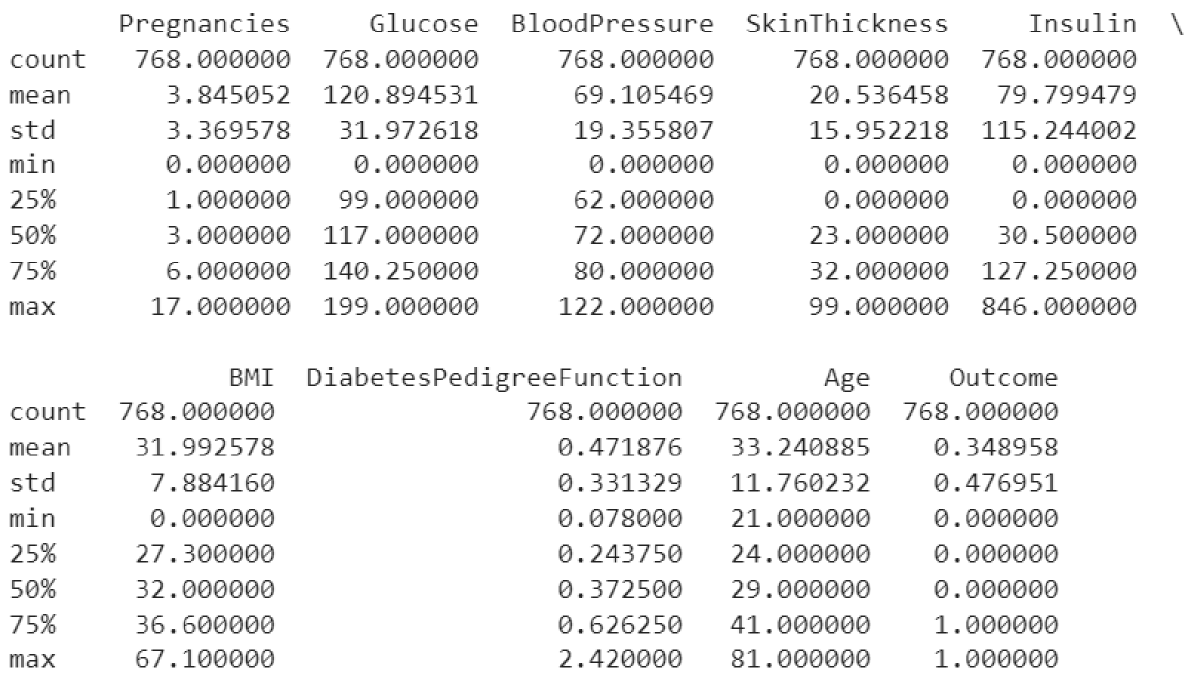
#Contains both the first and second parts of the project#

Detection of Diabetes Using Deep Learning

Diabetes is a metabolic disorder that occurs when the body is unable to produce a sufficient amount of insulin or effectively utilize it, leading to elevated levels of blood sugar. Early diagnosis and appropriate treatment can help control diabetes. In this context, the use of deep learning applications holds significant potential for effectively diagnosing diabetes and improving the treatment process. This research aims to develop a diabetes diagnostic model based on deep learning using clinical and physical data. To achieve this goal, various sampling techniques will be applied to the existing dataset, folloId by the application of traditional machine learning algorithms and methods such as Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), and complex neural network architectures for diabetes diagnosis. The performance evaluation of this study will utilize the widely used Pima Indians Diabetes dataset. The obtained results will be used to assess whether this deep learning-based model is an effective tool for diabetes diagnosis.

1. The dataset used and its review

In this project I developed for the diagnosis of diabetes, I use the Pima Indians Diabetes Dataset, which is widely used around the world. If I examine the mentioned dataset in general;



```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

Image 1.0

This dataset consists of a total of 768 observations, each representing an individual. Each observation includes information about a person's health characteristics and diabetes status. The features in the dataset are as follows:

1. Pregnancies: This feature represents the number of pregnancies a person has experienced. Its minimum value is 0 and its maximum value is 17.
2. Glucose: This feature expresses the glucose level in a person's blood in milligrams per deciliter (mg/dL).
3. Blood Pressure: This feature represents a person's blood pressure and is expressed in millimeters of mercury (mm Hg).
4. Skin Thickness: This feature represents the thickness of a person's skin in millimeters.
5. Insulin: This feature represents a person's insulin level and is expressed in microunits per milliliter (MicroU/mL).
6. BMI (Body Mass Index): This feature represents a person's body mass index, calculated using the formula: Weight (kg) / Height (m)^2.
7. Diabetes Pedigree Function: This feature represents the value of a function based on the patient's diabetes history in the family tree.
8. Age: This feature represents a person's age. The youngest person is 21 years old and the oldest person is 81 years old. The overall average is 33.240885.
9. Outcome: This feature indicates whether a person has diabetes or not. 0 represents the absence of diabetes, and 1 represents the presence of diabetes. Approximately 34.90% of participants in the dataset are patients

2. Application of Dataset Balancing Methods and Their Interactions

In machine learning applications, success is generally proportional to the quality and balance of the dataset on which the model is trained. Holver, real-world datasets often tend to be imbalanced, meaning that one class may have significantly more examples than another. This imbalance can negatively impact the model's training process and lead to misleading results. Therefore, in machine learning applications, resorting to dataset balancing strategies has become inevitable.

When I examine the distribution of the Outcome feature in our dataset by using BMI and Glucose characteristics as independent variables, the distribution of the Outcome feature is as follows;

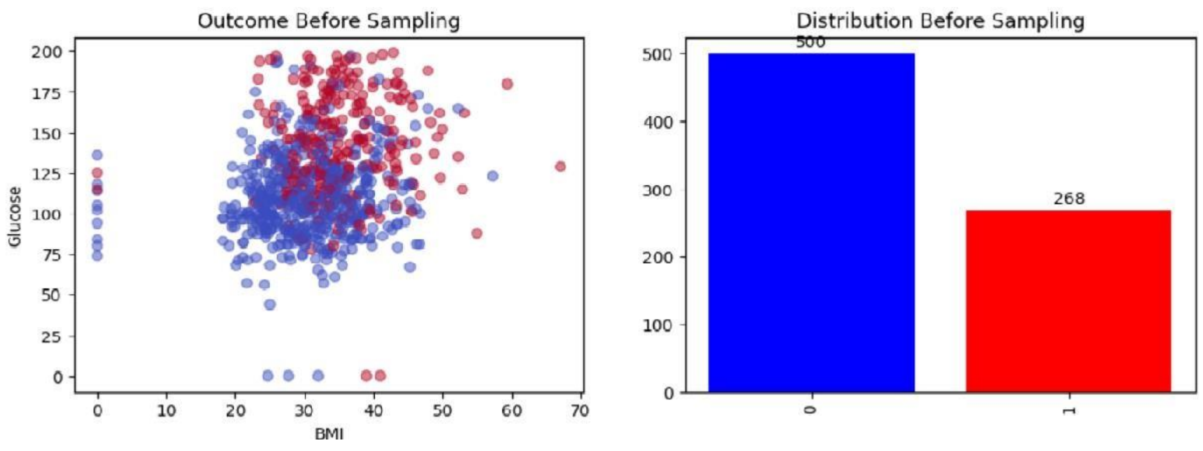
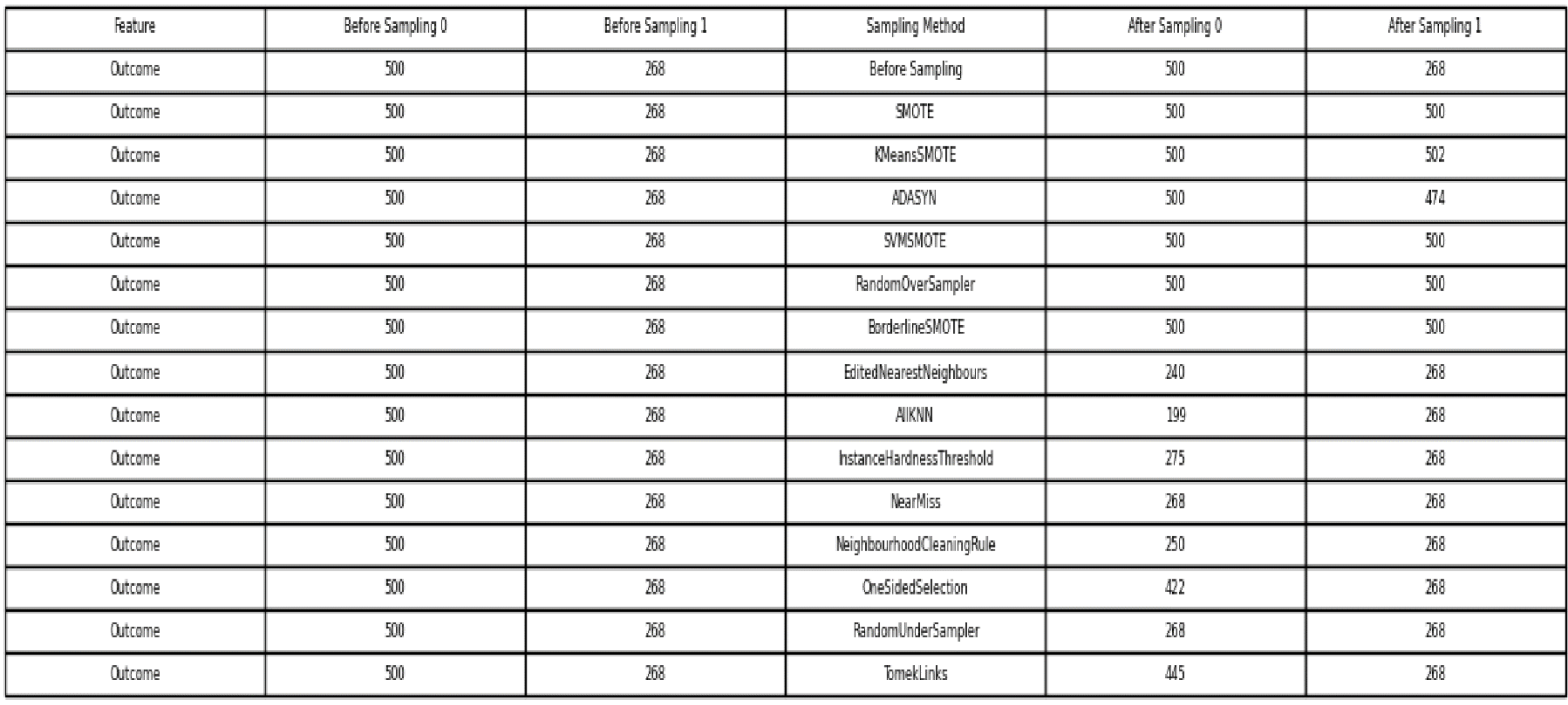


Image 2.0

As seen in the image 2.0, there is an imbalance in the data distribution in our outcome feature.



Feature	Before Sampling 0	Before Sampling 1	Sampling Method	After Sampling 0	After Sampling 1
Outcome	500	268	Before Sampling	500	268
Outcome	500	268	SMOTE	500	500
Outcome	500	268	KMeansSMOTE	500	502
Outcome	500	268	ADASYN	500	474
Outcome	500	268	SVMSMOTE	500	500
Outcome	500	268	RandomOverSampler	500	500
Outcome	500	268	BorderlineSMOTE	500	500
Outcome	500	268	EditedNearestNeighbours	240	268
Outcome	500	268	AllKNN	199	268
Outcome	500	268	InstanceHardnessThreshold	275	268
Outcome	500	268	NearMiss	268	268
Outcome	500	268	NeighbourhoodCleaningRule	250	268
Outcome	500	268	OneSidedSelection	422	268
Outcome	500	268	RandomUnderSampler	268	268
Outcome	500	268	TomekLinks	445	268

Image 2.1

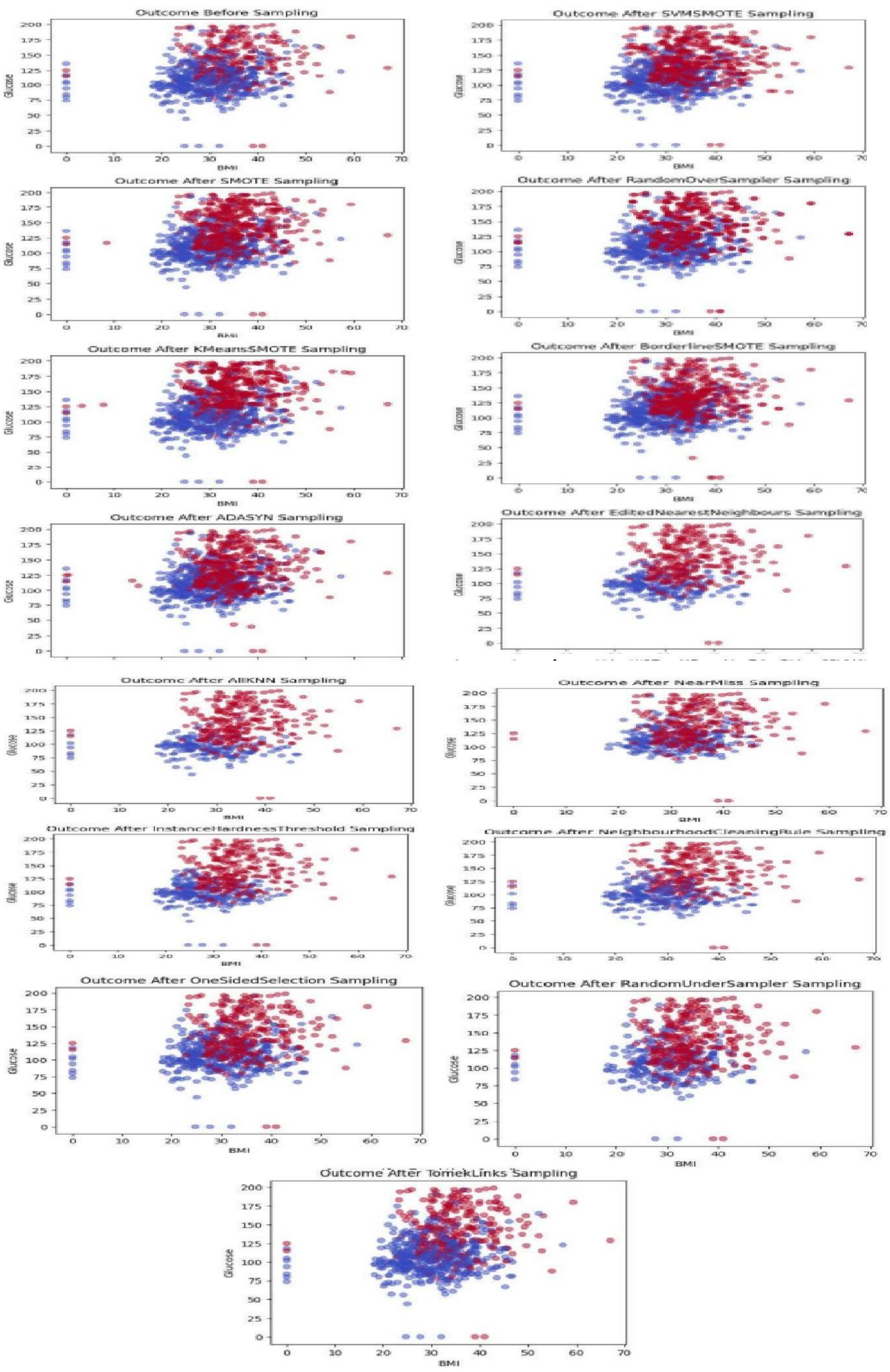

Outcome Before Sampling
Outcome After SVMSMOTE Sampling
Outcome After SMOTE Sampling
Outcome After RandomOverSampler Sampling
Outcome After KMeansSMOTE Sampling
Outcome After BorderlineSMOTE Sampling
Outcome After ADASYN Sampling
Outcome After EditedNearestNeighbours Sampling
Outcome After AllKNN Sampling
Outcome After NearMiss Sampling
Outcome After InstanceHardnessThreshold Sampling
Outcome After NeighbourhoodCleaningRule Sampling
Outcome After OneSidedSelection Sampling
Outcome After RandomUnderSampler Sampling
Outcome After TomekLinks Sampling
Glucose
BMI

Image 2.2

When methods for correcting class imbalance Ire applied, the distribution of classes 0 and 1 acquired a structure similar to the example in Figure 2.1. In the subsequent stage, the new distributions of the outcome feature, as seen in Figure 2.2, Ire examined for each applied method, utilizing information derived from BMI and Glucose features. It was observed that the relevant methods generally did not lead to information loss.

3. Deciding whether to build a pipeline or not

We Ire wondering how favorable this situation would be for us when I pipelined an oversample and an undersample method. To understand this situation, I applied the random forest classification method for the balanced datasets I obtained before, and then applied the randomforest classification method again by pipelined the oversample and undersample methods with the highest machine learning metric values.

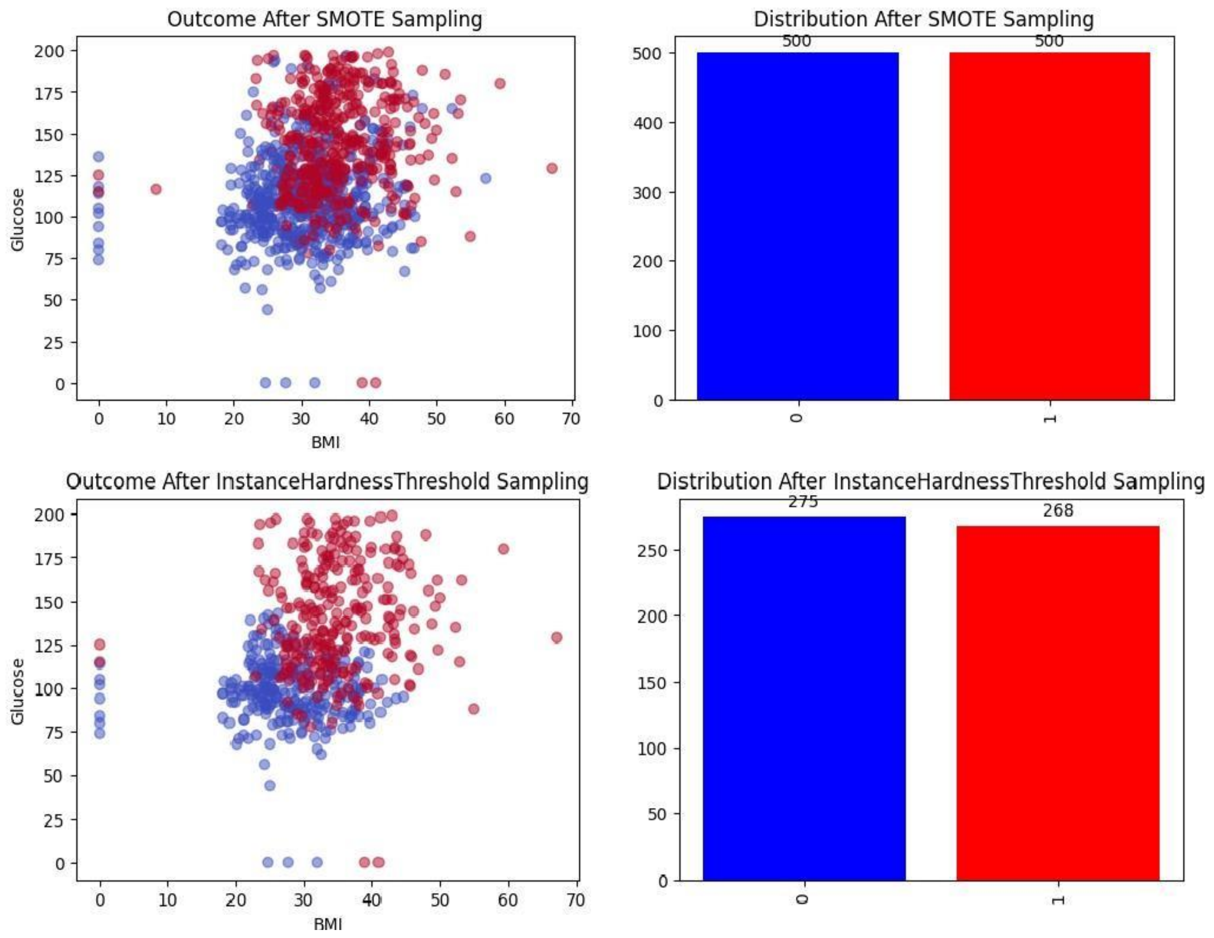


Image 3.0

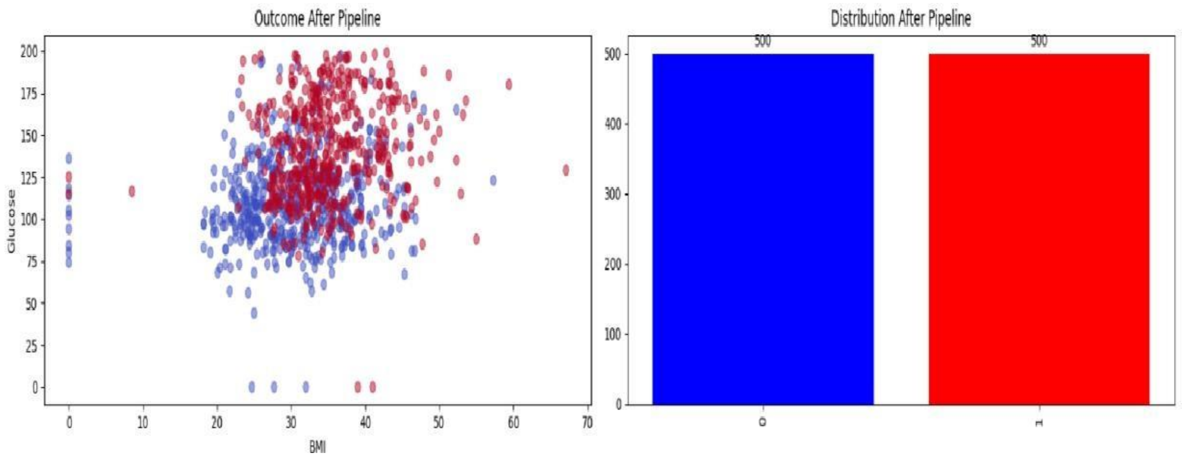


Image 3.1

While the accuracy value for Smote was 0.815 and the accuracy value for Instance Hardness Threshold was 0.92660, I saw that the accuracy value decreased to 0.76 when I pipelined the two. The fact that there is no guarantee that our pipeline application will cause an increase in machine learning metrics and that I have already implemented 14 different data balance methods made us decide not to pipeline.

4. Traditional Machine Learning Algorithms

This section includes classification studies carried out on the PIMA diabetes data set with 4 different machine learning techniques (KNN, Random Forest, XGBoost, LightGBM) for the early diagnosis of diabetes. The main purpose of classification studies is to increase prediction accuracy. In this study, 14 different resampling methods Ire used on the data set other than the original data set to increase the success of the classifiers. For each machine learning model, 60 operations Ire performed without sampling and with resampling.

We used 4 different metrics to examine each classification method I applied, these are; accuracy, precision, recall and f1 score.

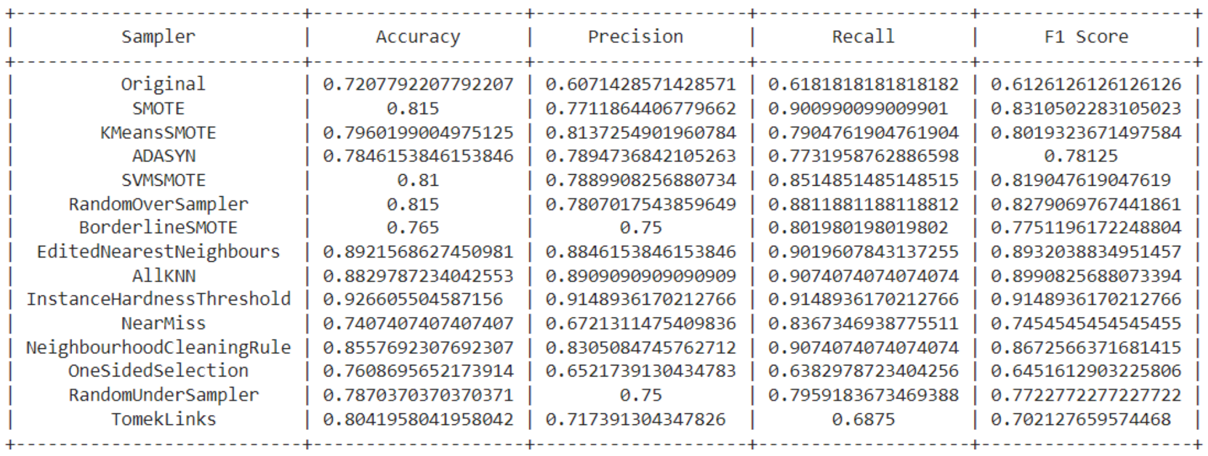
Accuracy: Specifies the proportion of the total number of samples that the model predicts correctly. High accuracy indicates that the overall performance of the model is good.

Precision: Specifies the probability that the samples predicted as positive by the model are actually positive. High precision indicates that the model is less likely to make false positive predictions.

Recall : Indicates how much of the truly positive samples are correctly detected by the model. A high recall indicates that the model is not missing true positives.

F1 Score: It is the harmonic average of precision and recall values. Provides a balanced measure of performance. A high F1 score indicates a model with high both precision and recall.

1. For Random Forest Classification



```
+---------------------------+--------------------+--------------------+--------------------+--------------------+
|          Sampler          |      Accuracy      |     Precision      |       Recall       |      F1 Score      |
+---------------------------+--------------------+--------------------+--------------------+--------------------+
|          Original         | 0.7207792207792207 | 0.6071428571428571 | 0.6181818181818182 | 0.6126126126126126 |
|           SMOTE           |       0.815        | 0.7711864406779662 | 0.900990099009901  | 0.8310502283105023 |
|        KMeansSMOTE        | 0.7960199004975125 | 0.8137254901960784 | 0.7904761904761904 | 0.8019323671497584 |
|           ADASYN          | 0.7846153846153846 | 0.7894736842105263 | 0.7731958762886598 |      0.78125       |
|          SVMSMOTE         |        0.81        | 0.7889908256880734 | 0.8514851485148515 | 0.819047619047619  |
|     RandomOverSampler     |       0.815        | 0.7807017543859649 | 0.8811881188118812 | 0.8279069767441861 |
|      BorderlineSMOTE      |       0.765        |        0.75        | 0.801980198019802  | 0.7751196172248804 |
|  EditedNearestNeighbours  | 0.8921568627450981 | 0.8846153846153846 | 0.9019607843137255 | 0.8932038834951457 |
|           AllKNN          | 0.8829787234042553 | 0.9090909090909090 | 0.9074074074074074 | 0.8990825688073394 |
|  InstanceHardnessThreshold| 0.926605504587156  | 0.9148936170212766 | 0.9148936170212766 | 0.9148936170212766 |
|          NearMiss         | 0.7407407407407407 | 0.6721311475409836 | 0.8367346938775511 | 0.7454545454545455 |
| NeighbourhoodCleaningRule | 0.8557692307692307 | 0.8305084745762712 | 0.9074074074074074 | 0.8672566371681415 |
|     OneSidedSelection     | 0.7608695652173914 | 0.6521739130434783 | 0.6382978723404256 | 0.6451612903225806 |
|     RandomUnderSampler    | 0.7870370370370371 |        0.75        | 0.7959183673469388 | 0.7722772277227722 |
|         TomekLinks        | 0.8041958041958042 | 0.717391304347826  |       0.6875       | 0.702127659574468  |
+---------------------------+--------------------+--------------------+--------------------+--------------------+
```

Image 4.1

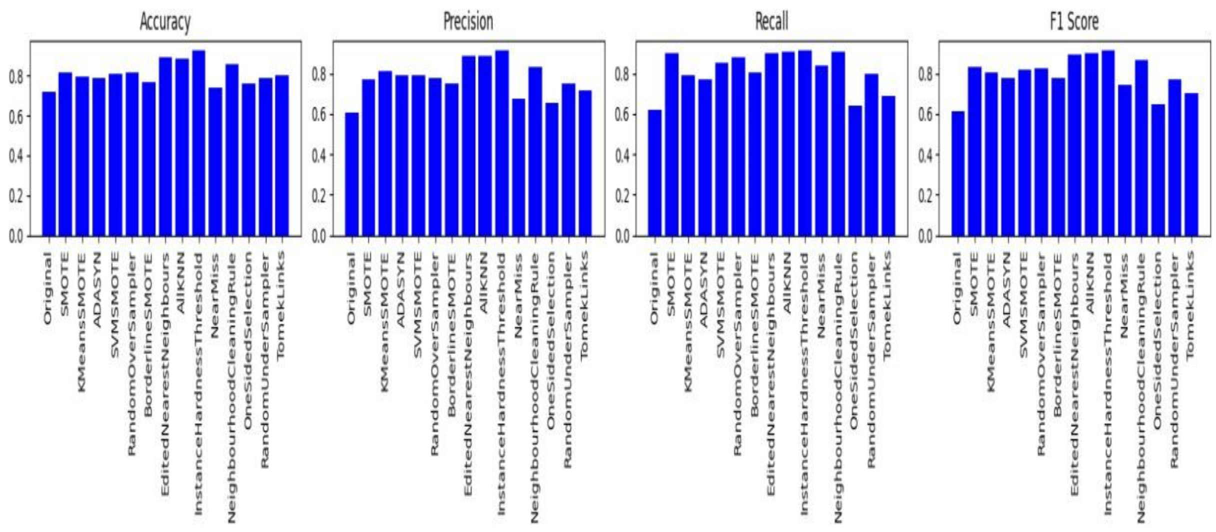
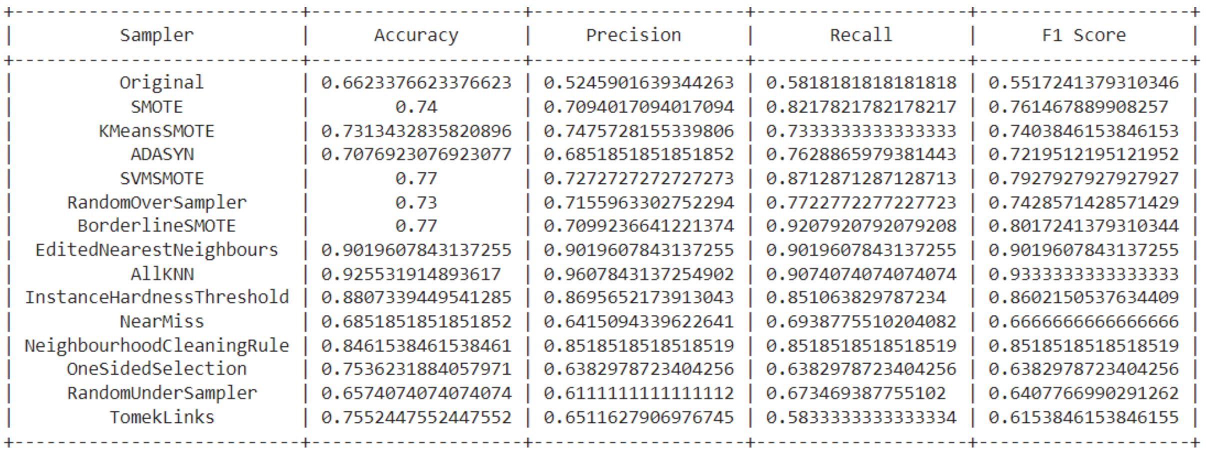


Image 4.2

In general, when the Random Forest classification method is applied, the prominent dataset balancing methods are: EditedNearestNeighbours, AllKNN and InstanceHardnessThreshold. The InstanceHardnessThreshold method has higher accuracy (0.9266) and precision (0.9148) values compared to others. On the other hand, the EditedNearestNeighbours method focuses on higher recall value. The AllKNN method stands out by providing a balance betIen these two features.

2. For KNN Classification



```
+---------------------------+--------------------+--------------------+--------------------+--------------------+
|          Sampler          |      Accuracy      |     Precision      |       Recall       |      F1 Score      |
+---------------------------+--------------------+--------------------+--------------------+--------------------+
|         Original          | 0.6623376623376623 | 0.5245901639344263 | 0.5818181818181818 | 0.5517241379310346 |
|           SMOTE           |        0.74        | 0.7094017094017094 | 0.8217821782178217 | 0.761467889908257  |
|        KMeansSMOTE        | 0.7313432835820896 | 0.7475728155339806 | 0.7333333333333333 | 0.7403846153846153 |
|          ADASYN           | 0.7076923076923077 | 0.6851851851851852 | 0.7628865979381443 | 0.7219512195121952 |
|         SVMSMOTE          |        0.77        | 0.7272727272727273 | 0.8712871287128713 | 0.7927927927927927 |
|     RandomOverSampler     |        0.73        | 0.7155963302752294 | 0.7722772277227723 | 0.7428571428571429 |
|      BorderlineSMOTE      |        0.77        | 0.7099236641221374 | 0.9207920792079208 | 0.8017241379310344 |
|  EditedNearestNeighbours  | 0.9019607843137255 | 0.9019607843137255 | 0.9019607843137255 | 0.9019607843137255 |
|          AllKNN           | 0.925531914893617  | 0.9607843137254902 | 0.9074074074074074 | 0.9333333333333333 |
| InstanceHardnessThreshold | 0.8807339449541285 | 0.8695652173913043 | 0.851063829787234  | 0.8602150537634409 |
|         NearMiss          | 0.6851851851851852 | 0.6415094339622641 | 0.6938775510204082 | 0.6666666666666666 |
| NeighbourhoodCleaningRule | 0.8461538461538461 | 0.8518518518518519 | 0.8518518518518519 | 0.8518518518518519 |
|     OneSidedSelection     | 0.7536231884057971 | 0.6382978723404256 | 0.6382978723404256 | 0.6382978723404256 |
|    RandomUnderSampler     | 0.6574074074074074 | 0.6111111111111112 | 0.673469387755102  | 0.6407766990291262 |
|        TomekLinks         | 0.7552447552447552 | 0.6511627906976745 | 0.5833333333333334 | 0.6153846153846155 |
+---------------------------+--------------------+--------------------+--------------------+--------------------+
```

Image 4.3

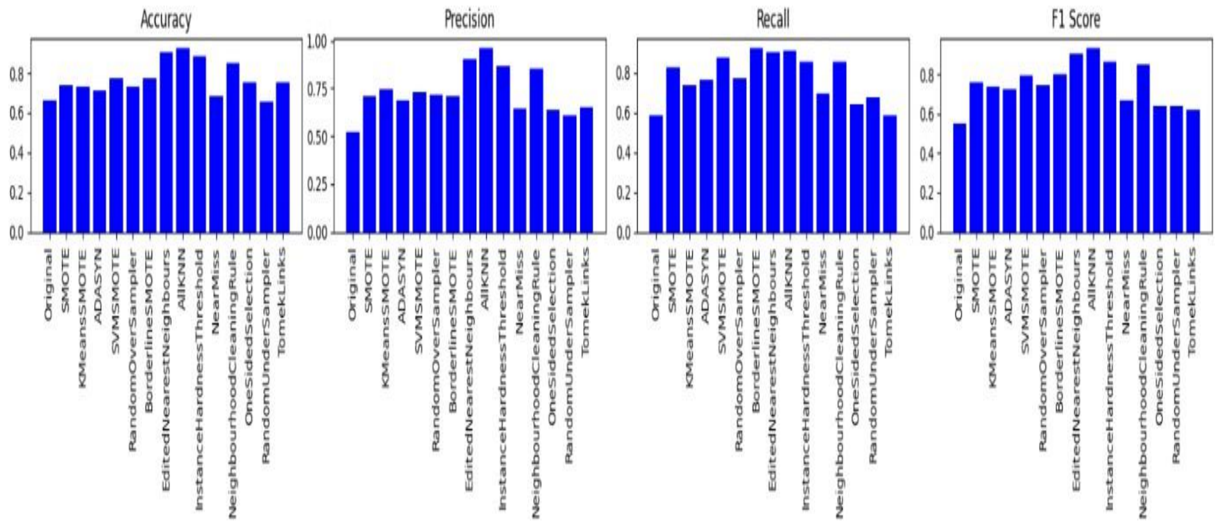


Image 4.4

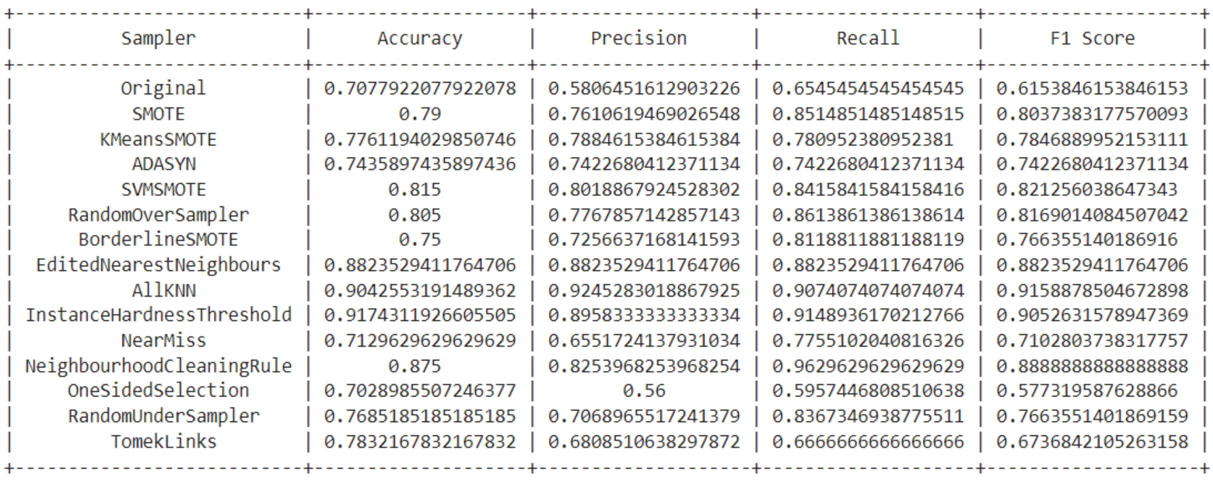
In general, when the KNN classification method is applied, the prominent dataset balancing methods are: EditedNearestNeighbours, AllKNN and InstanceHardnessThreshold.

Among these, the EditedNearestNeighbours method exhibits consistent performance across accuracy (0.9019), precision (0.9019), and recall (0.9019). Notably, it emphasizes a Ill-rounded approach to classification.

On the other hand, the AllKNN method demonstrates superior accuracy (0.9255) and precision (0.9608) values, showcasing its effectiveness in correctly identifying instances. It also strikes a balance with recall (0.9074), making it a robust choice for classification tasks.

In comparison, the InstanceHardnessThreshold method, while achieving slightly loIr accuracy (0.8807) and precision (0.8696) values, excels in instances where recall (0.8511) is crucial.

3. For xgboost Classification



```
+-------------------------+--------------------+--------------------+--------------------+--------------------+
|         Sampler         |      Accuracy      |     Precision      |       Recall       |      F1 Score      |
+-------------------------+--------------------+--------------------+--------------------+--------------------+
|         Original        | 0.7077922077922078 | 0.5806451612903226 | 0.6545454545454545 | 0.6153846153846153 |
|          SMOTE          |        0.79        | 0.7610619469026548 | 0.8514851485148515 | 0.8037383177570093 |
|       KMeansSMOTE       | 0.7761194029850746 | 0.7884615384615384 | 0.780952380952381  | 0.7846889952153111 |
|          ADASYN         | 0.7435897435897436 | 0.7422680412371134 | 0.7422680412371134 | 0.7422680412371134 |
|         SVMSMOTE        |       0.815        | 0.8018867924528302 | 0.8415841584158416 | 0.821256038647343  |
|    RandomOverSampler    |       0.805        | 0.7767857142857143 | 0.8613861386138614 | 0.8169014084507042 |
|     BorderlineSMOTE     |        0.75        | 0.7256637168141593 | 0.8118811881188119 | 0.766355140186916  |
|  EditedNearestNeighbours | 0.8823529411764706 | 0.8823529411764706 | 0.8823529411764706 | 0.8823529411764706 |
|          AllKNN         | 0.9042553191489362 | 0.9245283018867925 | 0.9074074074074074 | 0.9158878504672898 |
| InstanceHardnessThreshold | 0.9174311926605505 | 0.8958333333333334 | 0.9148936170212766 | 0.9052631578947369 |
|         NearMiss        | 0.7129629629629629 | 0.6551724137931034 | 0.7755102040816326 | 0.7102803738317757 |
| NeighbourhoodCleaningRule |       0.875        | 0.8253968253968254 | 0.9629629629629629 | 0.8888888888888888 |
|     OneSidedSelection    | 0.7028985507246377 |        0.56        | 0.5957446808510638 | 0.577319587628866  |
|    RandomUnderSampler    | 0.7685185185185185 | 0.7068965517241379 | 0.8367346938775511 | 0.7663551401869159 |
|        TomekLinks       | 0.7832167832167832 | 0.6808510638297872 | 0.6666666666666666 | 0.6736842105263158 |
+-------------------------+--------------------+--------------------+--------------------+--------------------+
```

Image 4.5

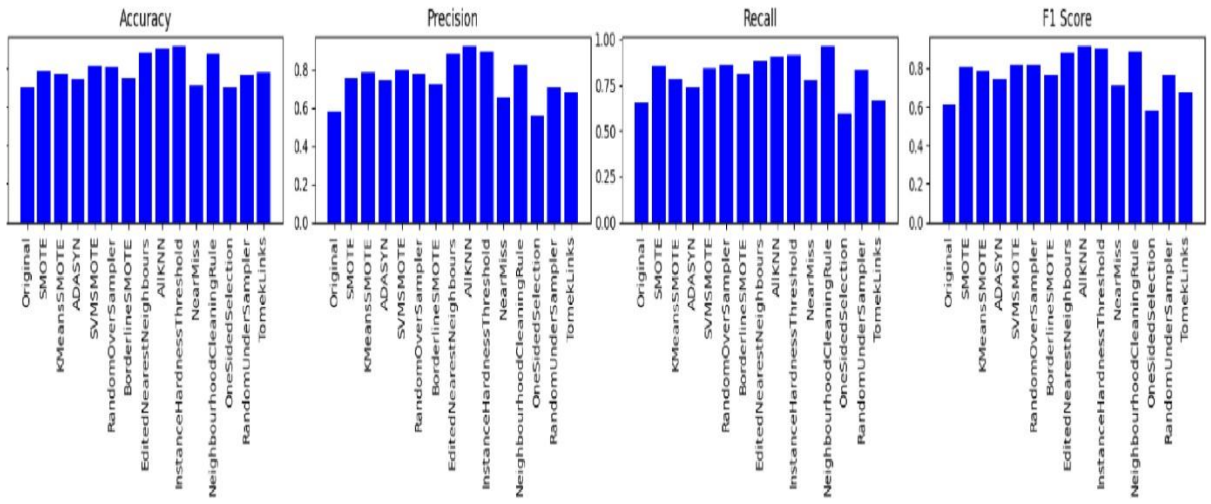


Image 4.6

In general, when the xgboost classification method is applied, the prominent dataset balancing methods are: EditedNearestNeighbours, AllKNN and InstanceHardnessThreshold.

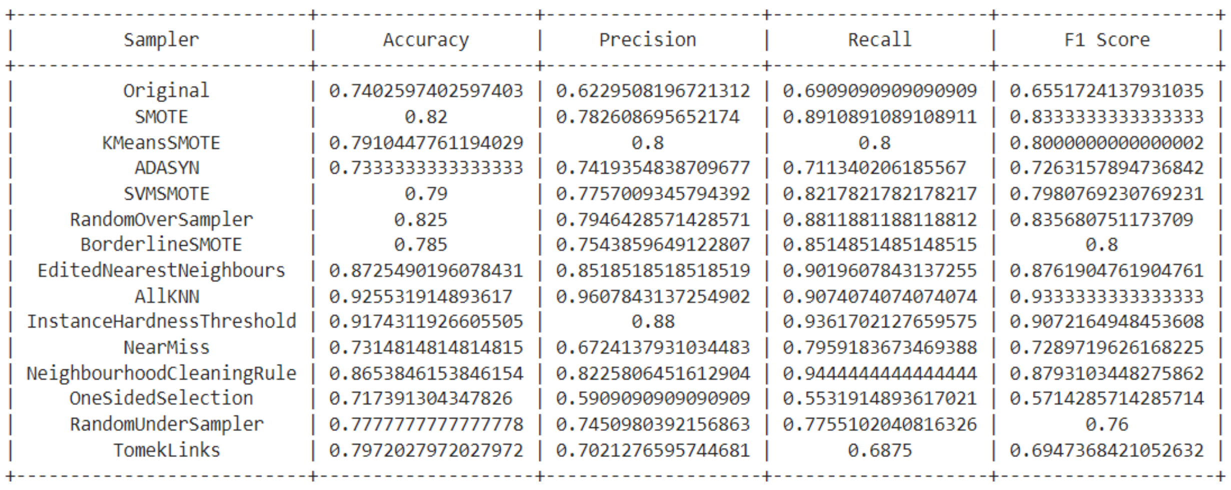
EditedNearestNeighbours consistently performs Ill across accuracy (0.8824), precision (0.8824), and recall (0.8824), showcasing a balanced and reliable classification approach.

In contrast, AllKNN stands out with superior accuracy (0.9043) and precision (0.9245), making it effective in correctly identifying instances. It maintains a commendable balance with recall (0.9074), presenting a robust choice for tasks prioritizing both precision and recall.

InstanceHardnessThreshold, while slightly loIr in accuracy (0.9174) and precision (0.8958), excels in scenarios where recall (0.9149) is crucial. It demonstrates strength in capturing instances significant for the classification task.

In summary, EditedNearestNeighbours is consistently balanced, AllKNN excels in accuracy and precision, and InstanceHardnessThreshold is particularly strong in recall-focused scenarios.

4. For lightgbm Classification



```
+-----------------------------+--------------------+--------------------+--------------------+--------------------+
|           Sampler           |      Accuracy      |     Precision      |       Recall       |      F1 Score      |
+-----------------------------+--------------------+--------------------+--------------------+--------------------+
|           Original          | 0.7402597402597403 | 0.6229508196721312 | 0.6909090909090909 | 0.6551724137931035 |
|            SMOTE            |        0.82        | 0.782608695652174  | 0.8910891089108911 | 0.8333333333333333 |
|         KMeansSMOTE         | 0.7910447761194029 |        0.8         |        0.8         | 0.8000000000000002 |
|            ADASYN           | 0.7333333333333333 | 0.7419354838709677 | 0.711340206185567  | 0.7263157894736842 |
|           SVMSMOTE          |        0.79        | 0.7757009345794392 | 0.8217821782178217 | 0.7980769230769231 |
|      RandomOverSampler      |       0.825        | 0.7946428571428571 | 0.8811881188118812 | 0.835680751173709  |
|       BorderlineSMOTE       |       0.785        | 0.7543859649122807 | 0.8514851485148515 |        0.8         |
|   EditedNearestNeighbours   | 0.8725490196078431 | 0.8518518518518519 | 0.9019607843137255 | 0.8761904761904761 |
|            AllKNN           | 0.925531914893617  | 0.9607843137254902 | 0.9074074074074074 | 0.9333333333333333 |
|  InstanceHardnessThreshold  | 0.9174311926605505 |        0.88        | 0.9361702127659575 | 0.9072164948453608 |
|           NearMiss          | 0.7314814814814815 | 0.6724137931034483 | 0.7959183673469388 | 0.7289719626168225 |
|  NeighbourhoodCleaningRule  | 0.8653846153846154 | 0.8225806451612904 | 0.9444444444444444 | 0.8793103448275862 |
|      OneSidedSelection      | 0.717391304347826  | 0.5909090909090909 | 0.5531914893617021 | 0.5714285714285714 |
|      RandomUnderSampler     | 0.7777777777777778 | 0.7450980392156863 | 0.7755102040816326 |        0.76        |
|          TomekLinks         | 0.7972027972027972 | 0.7021276595744681 |       0.6875       | 0.6947368421052632 |
+-----------------------------+--------------------+--------------------+--------------------+--------------------+
```

Image 4.7

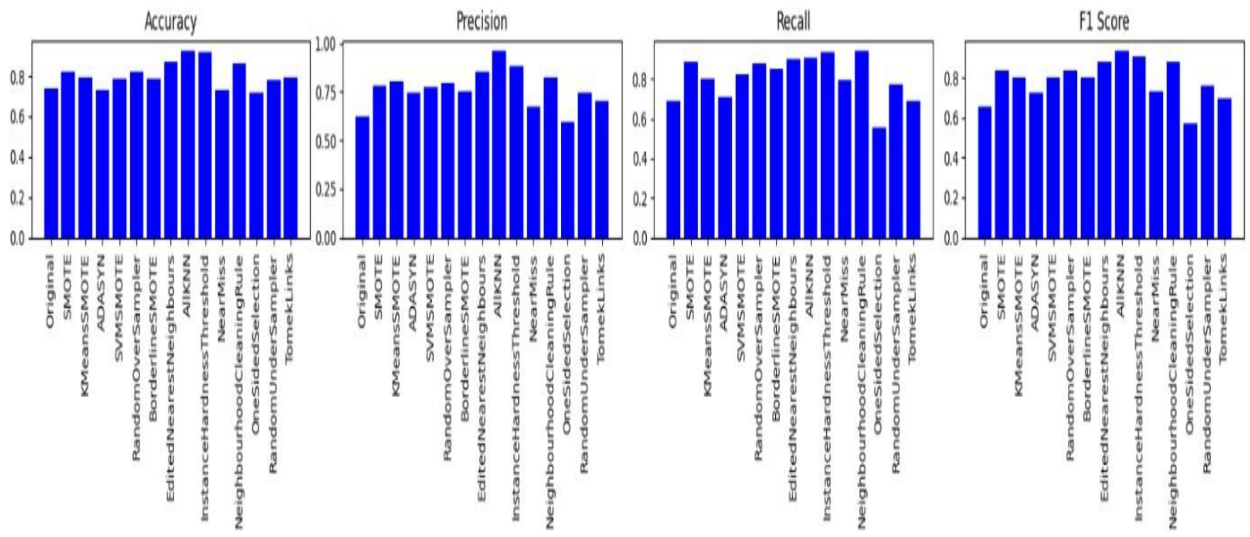


Image 4.8

In general, when the lightgbm classification method is applied, the prominent dataset balancing methods are: EditedNearestNeighbours, AllKNN, InstanceHardnessThreshold and NeighbourhoodCleaningRule.

EditedNearestNeighbours has balanced performance across accuracy(0.87), precision(0.85), recall(0.90), and F1 score(0.87).

AllKNN stands out with high accuracy(0.92) and precision(0.96), effectively identifying instances while maintaining commendable balance with recall.

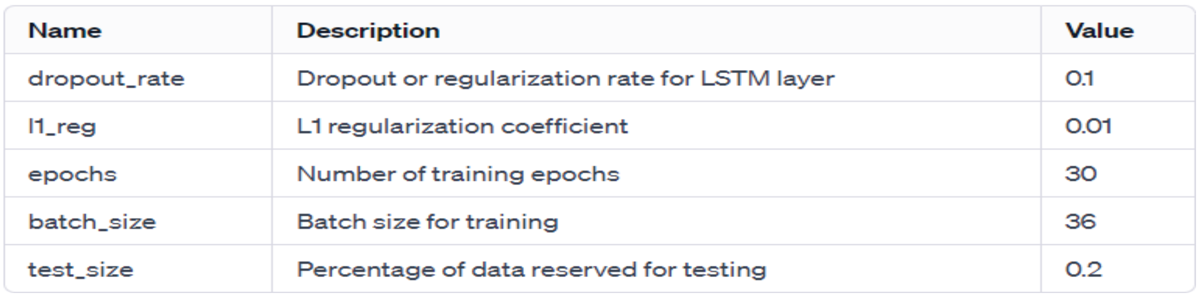
InstanceHardnessThreshold is slightly loIr in accuracy(0.91) and precision(0.88) but excels in recall-focused scenarios, showcasing strength in capturing significant instances.

NeighbourhoodCleaningRule has balanced performance with slightly loIr accuracy(0.86) and precision(0.82) compared to other methods, offering trade-offs.

5. Implementation of LSTM

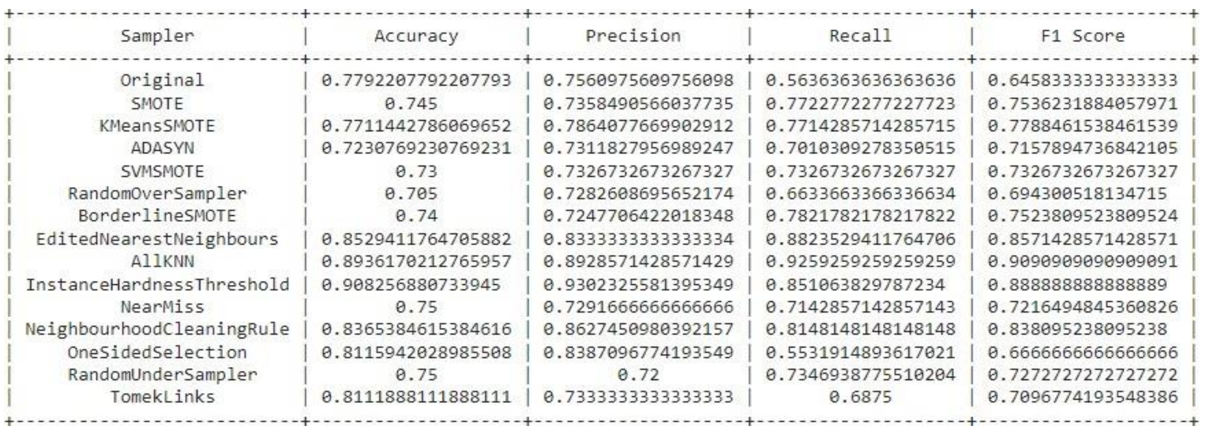
LSTM, which stands for "Long Short-Term Memory," is a type of recurrent neural network (RNN). Recurrent neural networks are structures that can remember information from previous steps and utilize this information in future steps. Holver, it is known that standard RNNs struggle to handle long-term dependencies and encounter an issue called the "vanishing gradient problem." LSTM is a type of RNN designed to overcome such problems.

The ability to integrate different types of data such as BMI, glucose, age, etc., and model complex relationships has led us to implement LSTM (Long Short-Term Memory) due to its advantage in addressing long-term dependencies. LSTM allows for a more effective capture of long-term factors influencing the development and progression of the disease.



Name	Description	Value
dropout_rate	Dropout or regularization rate for LSTM layer	0.1
l1_reg	L1 regularization coefficient	0.01
epochs	Number of training epochs	30
batch_size	Batch size for training	36
test_size	Percentage of data reserved for testing	0.2

Image 5.1



```
+----------------------------+---------------------+---------------------+---------------------+---------------------+
|          Sampler           |      Accuracy       |      Precision      |        Recall       |       F1 Score      |
+----------------------------+---------------------+---------------------+---------------------+---------------------+
|          Original          | 0.7792207792207793  | 0.7560975609756098  | 0.5636363636363636  | 0.6458333333333333  |
|           SMOTE            |        0.745        | 0.7358490566037735  | 0.7722772277227723  | 0.7536231884057971  |
|        KMeansSMOTE         | 0.7711442786069652  | 0.7864077669902912  | 0.7714285714285715  | 0.7788461538461539  |
|           ADASYN           | 0.7230769230769231  | 0.7311827956989247  | 0.7010309278350515  | 0.7157894736842105  |
|          SVMSMOTE          |         0.73        | 0.7326732673267327  | 0.7326732673267327  | 0.7326732673267327  |
|     RandomOverSampler      |        0.705        | 0.7282608695652174  | 0.6633663366336634  |  0.694300518134715  |
|      BorderlineSMOTE       |         0.74        | 0.7247706422018348  | 0.7821782178217822  | 0.7523809523809524  |
|  EditedNearestNeighbours   | 0.8529411764705882  | 0.8333333333333334  | 0.8823529411764706  | 0.8571428571428571  |
|           AllKNN           | 0.8936170212765957  | 0.8928571428571429  | 0.9259259259259259  | 0.9090909090909091  |
| InstanceHardnessThreshold  |  0.908256880733945  | 0.9302325581395349  |  0.851063829787234  |  0.888888888888889  |
|          NearMiss          |         0.75        | 0.7291666666666666  | 0.7142857142857143  | 0.7216494845360826  |
| NeighbourhoodCleaningRule  | 0.8365384615384616  | 0.8627450980392157  | 0.8148148148148148  |  0.838095238095238  |
|     OneSidedSelection      | 0.8115942028985508  | 0.8387096774193549  | 0.5531914893617021  | 0.6666666666666666  |
|    RandomUnderSampler      |         0.75        |         0.72        | 0.7346938775510204  | 0.7272727272727272  |
|         TomekLinks         | 0.8111888111888111  | 0.7333333333333333  |        0.6875       | 0.7096774193548386  |
+----------------------------+---------------------+---------------------+---------------------+---------------------+
```

Image 5.2

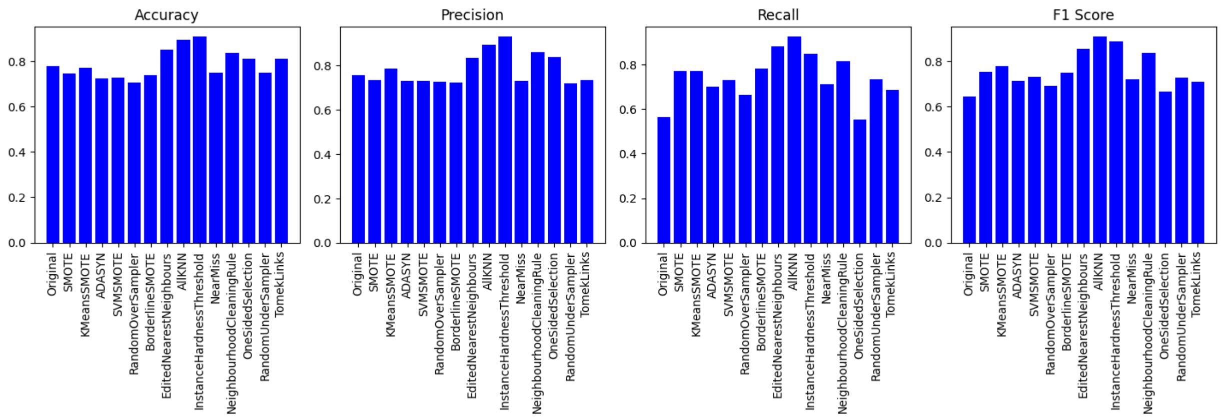


Image 5.3

A more concise explanation of the metrics obtained when examining the model's performance in a scenario where the InstanceHardnessThreshold (IHT) balancing method is used(best accuracy):

Accuracy: 0.91

It shows how much of the total samples the model correctly classified. High accuracy indicates that the overall performance of the model is good.

Precision: 0.93

It expresses the rate of examples that the model predicts as positive are actually positive. High precision indicates that positive predictions are reliable.

Recall (Sensitivity): 0.87

It measures how much of the samples it correctly classifies as truly positive. High sensitivity indicates that it does not tend to miss positive samples.

F1 Score: 0.90

As a metric that combines precision and sensitivity, it takes both false positives and false negatives into account. A high F1 score indicates a good balance achieved by the balancing method.

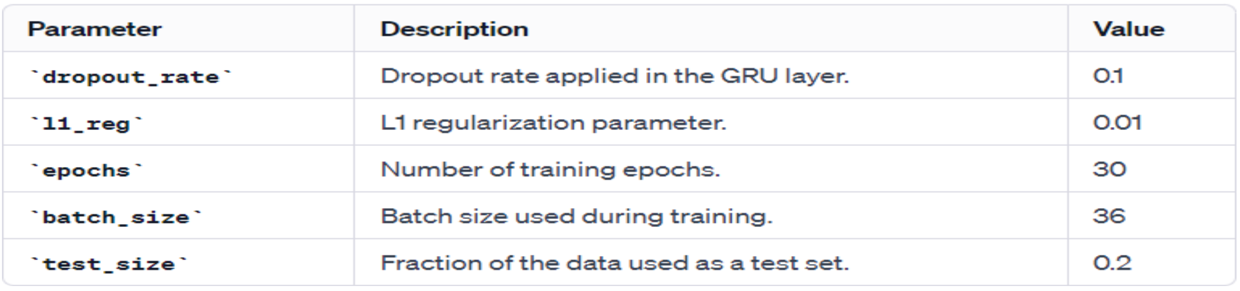
6. Implementation of GRU

In fact, LSTM and GRU are quite similar models. Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) are models of the recurrent neural network (RNN) type and are used to deal with sequential data such as time series. Both are designed to address long-term dependencies, which is the main challenge of RNN. Here are the main differences betlen GRU and LSTM:

1. Cell Structure:
• LSTM (Long Short-Term Memory): LSTM has a cell structure with three main gates (input, output and forgetting gates). These gates help control how much information is forgotten and how much is retained.
• GRU (Gated Recurrent Unit): GRU has a more simplified cell structure and contains only two gates: reset and update gates. Therefore, it has felr parameters than LSTM.

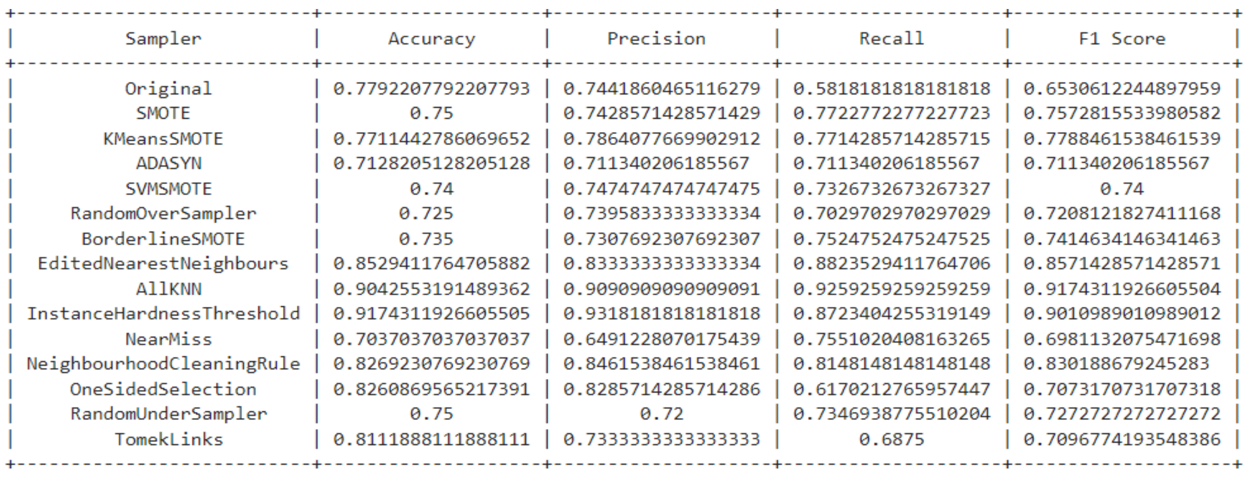
2. Number of Parameters:
• LSTM: LSTM contains more parameters because there are three gates in each cell.
• GRU: GRU contains felr parameters because the cell structure is simpler.

3. Training Duration:
• LSTM: LSTM may take longer to train because it contains more parameters.
• GRU: Since GRU contains felr parameters, the training time may be shorter.



Parameter	Description	Value
`dropout_rate`	Dropout rate applied in the GRU layer.	0.1
`l1_reg`	L1 regularization parameter.	0.01
`epochs`	Number of training epochs.	30
`batch_size`	Batch size used during training.	36
`test_size`	Fraction of the data used as a test set.	0.2

Image 6.1



```
+----------------------------+---------------------+---------------------+---------------------+---------------------+
|          Sampler           |      Accuracy       |      Precision      |        Recall       |       F1 Score      |
+----------------------------+---------------------+---------------------+---------------------+---------------------+
|          Original          | 0.7792207792207793  | 0.7441860465116279  | 0.5818181818181818  | 0.6530612244897959  |
|           SMOTE            |        0.75         | 0.7428571428571429  | 0.7722772277227723  | 0.7572815533980582  |
|        KMeansSMOTE         | 0.7711442786069652  | 0.7864077669902912  | 0.7714285714285715  | 0.7788461538461539  |
|           ADASYN           | 0.7128205128205128  |  0.711340206185567  |  0.711340206185567  |  0.711340206185567  |
|          SVMSMOTE          |        0.74         | 0.7474747474747475  | 0.7326732673267327  |        0.74         |
|     RandomOverSampler      |       0.725         | 0.7395833333333334  | 0.7029702970297029  | 0.7208121827411168  |
|      BorderlineSMOTE       |       0.735         | 0.7307692307692307  | 0.7524752475247525  | 0.7414634146341463  |
|  EditedNearestNeighbours   | 0.8529411764705882  | 0.8333333333333334  | 0.8823529411764706  | 0.8571428571428571  |
|           AllKNN           | 0.9042553191489362  | 0.9090909090909091  | 0.9259259259259259  | 0.9174311926605504  |
| InstanceHardnessThreshold  | 0.9174311926605505  | 0.9318181818181818  | 0.8723404255319149  | 0.9010989010989012  |
|          NearMiss          | 0.7037037037037037  | 0.6491228070175439  | 0.7551020408163265  | 0.6981132075471698  |
| NeighbourhoodCleaningRule  | 0.8269230769230769  | 0.8461538461538461  | 0.8148148148148148  |  0.830188679245283  |
|     OneSidedSelection      | 0.8260869565217391  | 0.8285714285714286  | 0.6170212765957447  | 0.7073170731707318  |
|     RandomUnderSampler     |        0.75         |        0.72         | 0.7346938775510204  | 0.7272727272727272  |
|         TomekLinks         | 0.8111888111888111  | 0.7333333333333333  |       0.6875        | 0.7096774193548386  |
+----------------------------+---------------------+---------------------+---------------------+---------------------+
```

Image 6.2

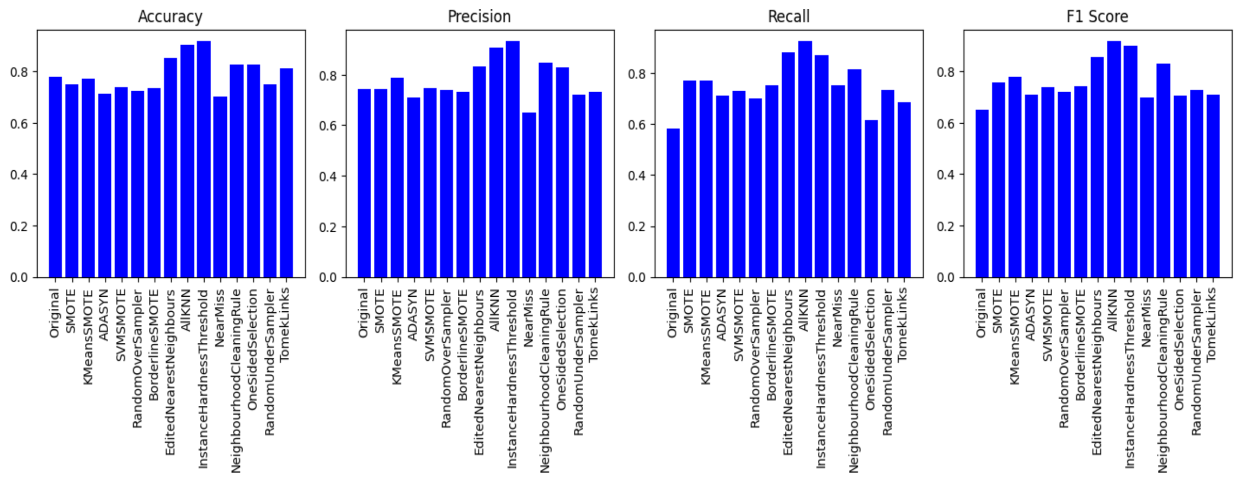


Image 6.3

A more concise explanation of the metrics obtained when examining the model's performance in a scenario where the InstanceHardnessThreshold (IHT) balancing method is used(best accuracy):

Accuracy: 0.91

It shows how much of the total samples the model correctly classified. High accuracy indicates that the overall performance of the model is good.

Precision: 0.93

It expresses the rate of examples that the model predicts as positive are actually positive. High precision indicates that positive predictions are reliable.

Recall (Sensitivity): 0.87

It measures how much of the samples it correctly classifies as truly positive. High sensitivity indicates that it does not tend to miss positive samples.

F1 Score: 0.90

As a metric that combines precision and sensitivity, it takes both false positives and false negatives into account. A high F1 score indicates a good balance achieved by the balancing method.

As you can see, our result is almost the same as what I got in LSTM.

7. Applying Ensembles

Ensemble learning is a strategy of combining a number of different learning models to obtain a more polrful and generalizable model. In this section, I aimed to create an ensemble model by bringing together different neural network architectures such as CNN (Convolutional Neural Network), LSTM (Long Short-Term Memory), and GRU (Gated Recurrent Unit).

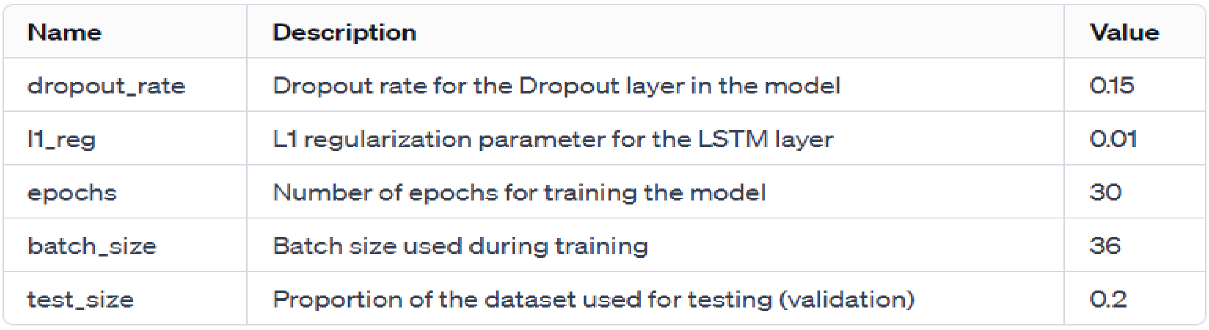
1. CNN + LSTM

Model Architecture:

- The neural network architecture consists of a combination of Conv1D (1D Convolutional) and LSTM layers.
- The data is reshaped to fit the Conv1D layer, which expects a 3D input with dimensions (samples, time steps, features).

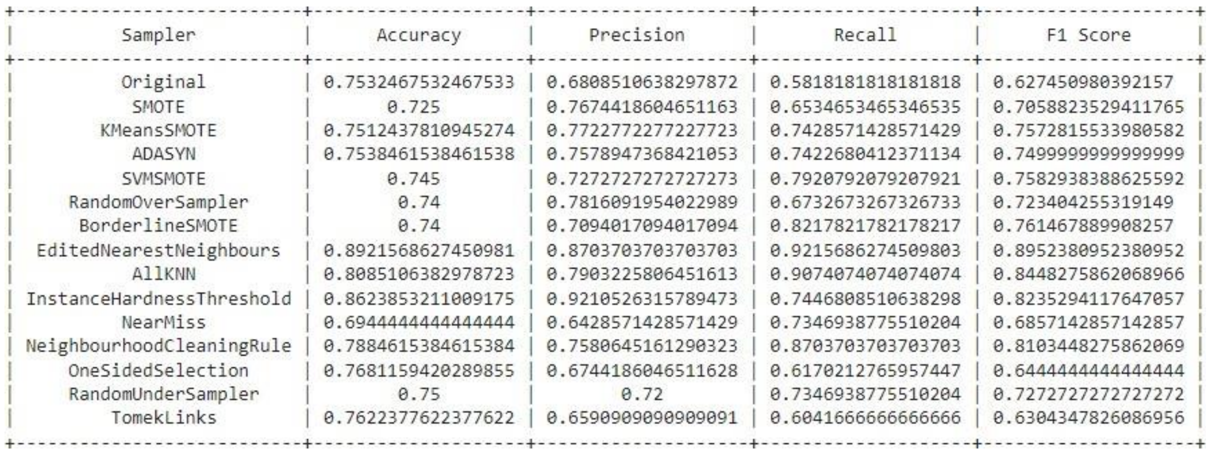
The model includes the following layers:

- Conv1D layer with 32 filters and a kernel size of 3, using ReLU activation.
- MaxPooling1D layer with a pool size of 2.
- LSTM layer with 50 units, a kernel regularization term (L1 regularization), and return sequences set to True.
- Dropout layer with a specified dropout rate.
- Flatten layer to convert the output to a one-dimensional array.
- Dense layer with 32 units and ReLU activation.
- Output layer with 1 unit and a sigmoid activation function for binary classification.



Name	Description	Value
dropout_rate	Dropout rate for the Dropout layer in the model	0.15
l1_reg	L1 regularization parameter for the LSTM layer	0.01
epochs	Number of epochs for training the model	30
batch_size	Batch size used during training	36
test_size	Proportion of the dataset used for testing (validation)	0.2

Image 7.1



```
+---------------------------+--------------------+--------------------+--------------------+--------------------+
|          Sampler          |      Accuracy      |     Precision      |       Recall       |      F1 Score      |
+---------------------------+--------------------+--------------------+--------------------+--------------------+
|          Original         | 0.7532467532467533 | 0.6808510638297872 | 0.5818181818181818 | 0.627450980392157  |
|           SMOTE           |       0.725        | 0.7674418604651163 | 0.6534653465346535 | 0.7058823529411765 |
|        KMeansSMOTE        | 0.7512437810945274 | 0.7722772277227723 | 0.7428571428571429 | 0.7572815533980582 |
|           ADASYN          | 0.7538461538461538 | 0.7578947368421053 | 0.7422680412371134 | 0.7499999999999999 |
|          SVMSMOTE         |       0.745        | 0.7272727272727273 | 0.7920792079207921 | 0.7582938388625592 |
|     RandomOverSampler     |        0.74        | 0.7816091954022989 | 0.6732673267326733 | 0.723404255319149  |
|      BorderlineSMOTE      |        0.74        | 0.7094017094017094 | 0.8217821782178217 | 0.761467889908257  |
|  EditedNearestNeighbours  | 0.8921568627450981 | 0.8703703703703703 | 0.9215686274509803 | 0.8952380952380952 |
|           AllKNN          | 0.8085106382978723 | 0.7903225806451613 | 0.9074074074074074 | 0.8448275862068966 |
| InstanceHardnessThreshold | 0.8623853211009175 | 0.9210526315789473 | 0.7446808510638298 | 0.8235294117647057 |
|          NearMiss         | 0.6944444444444444 | 0.6428571428571429 | 0.7346938775510204 | 0.6857142857142857 |
| NeighbourhoodCleaningRule | 0.7884615384615384 | 0.7580645161290323 | 0.8703703703703703 | 0.8103448275862069 |
|      OneSidedSelection    | 0.7681159420289855 | 0.6744186046511628 | 0.6170212765957447 | 0.6444444444444444 |
|     RandomUnderSampler    |        0.75        |        0.72        | 0.7346938775510204 | 0.7272727272727272 |
|         TomekLinks        | 0.7622377622377622 | 0.6590909090909091 | 0.6041666666666666 | 0.6304347826086956 |
+---------------------------+--------------------+--------------------+--------------------+--------------------+
```

Image 7.2

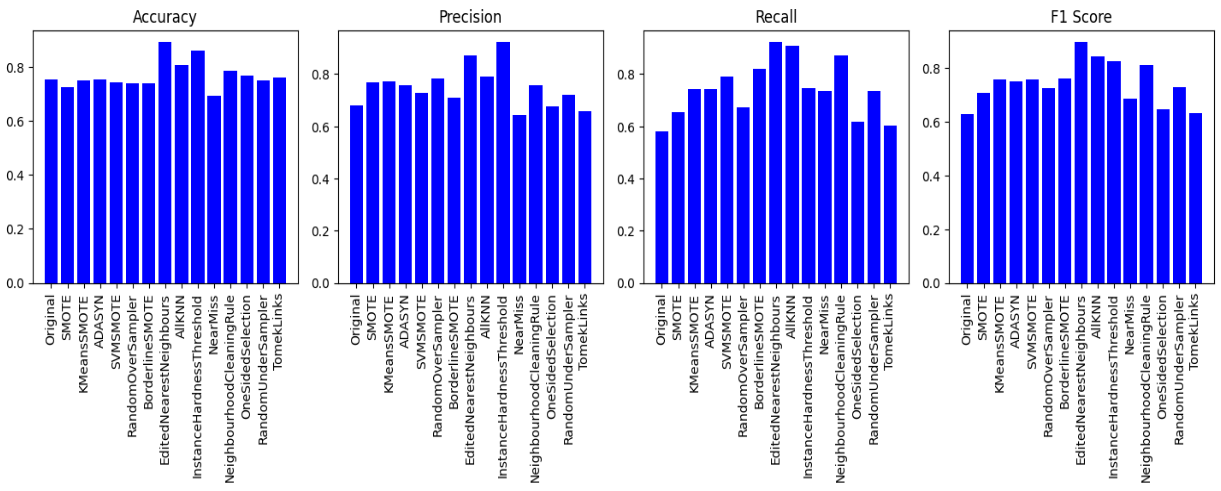


Image 7.3

As can be seen, the dataset balancing method with the highest machine learning metric values in the architecture I designed is EditedNearestNeighbours.

2. CNN + GRU

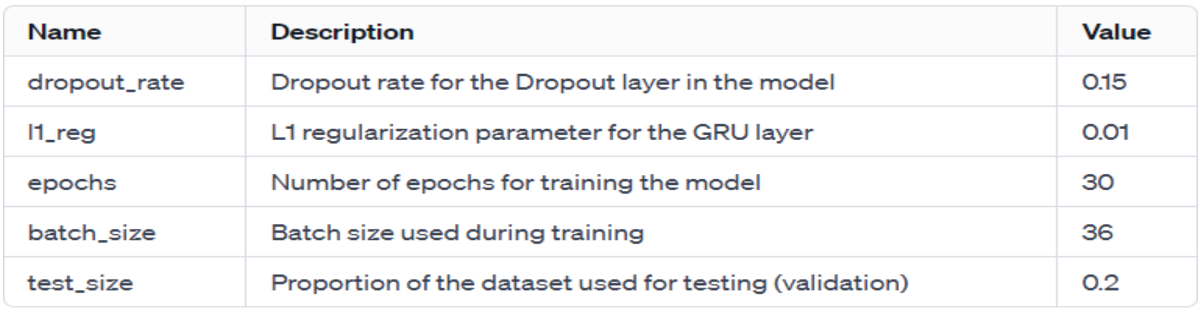
Model Architecture:

- The neural network architecture includes Conv1D and GRU layers for sequence modeling.

- The data is reshaped to fit the Conv1D layer, which expects a 3D input with dimensions (samples, time steps, features).

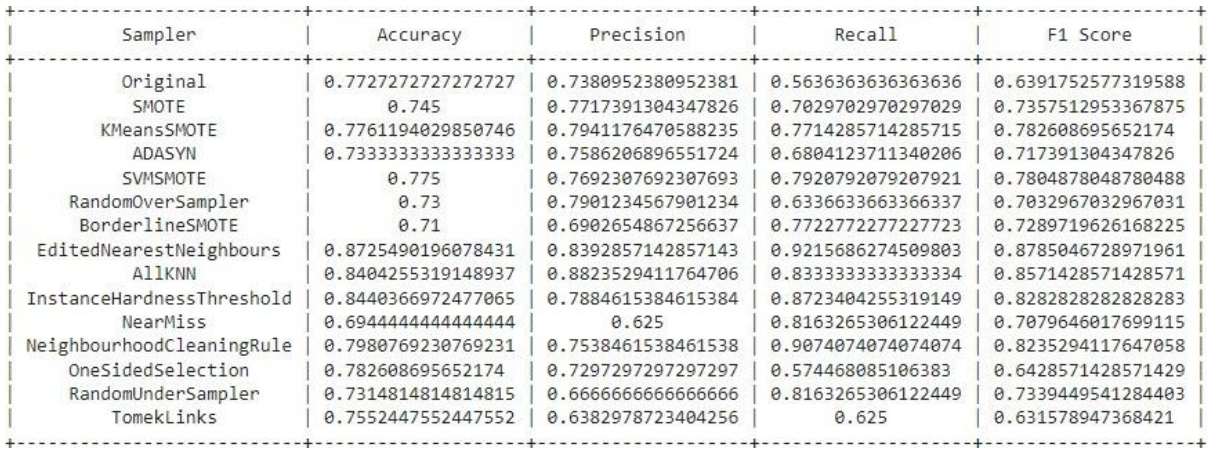
The model includes the following layers:

- Conv1D layer with 32 filters and a kernel size of 3, using ReLU activation.
- MaxPooling1D layer with a pool size of 2.
- GRU layer with 50 units, a kernel regularization term (L1 regularization), and return sequences set to True.
- Dropout layer with a specified dropout rate.
- Flatten layer to convert the output to a one-dimensional array.
- Dense layer with 32 units and ReLU activation.
- Output layer with 1 unit and a sigmoid activation function for binary classification.



Name	Description	Value
dropout_rate	Dropout rate for the Dropout layer in the model	0.15
l1_reg	L1 regularization parameter for the GRU layer	0.01
epochs	Number of epochs for training the model	30
batch_size	Batch size used during training	36
test_size	Proportion of the dataset used for testing (validation)	0.2

Image 7.4



```
+---------------------------+--------------------+--------------------+--------------------+--------------------+
|          Sampler          |      Accuracy      |     Precision      |       Recall       |      F1 Score      |
+---------------------------+--------------------+--------------------+--------------------+--------------------+
|         Original          | 0.7727272727272727 | 0.7380952380952381 | 0.5636363636363636 | 0.6391752577319588 |
|           SMOTE           |       0.745        | 0.7717391304347826 | 0.7029702970297029 | 0.7357512953367875 |
|        KMeansSMOTE        | 0.7761194029850746 | 0.7941176470588235 | 0.7714285714285715 | 0.782608695652174  |
|          ADASYN           | 0.7333333333333333 | 0.7586206896551724 | 0.6804123711340206 | 0.717391304347826  |
|         SVMSMOTE          |       0.775        | 0.7692307692307693 | 0.7920792079207921 | 0.7804878048780488 |
|     RandomOverSampler     |        0.73        | 0.7901234567901234 | 0.6336633663366337 | 0.7032967032967031 |
|      BorderlineSMOTE      |        0.71        | 0.6902654867256637 | 0.7722772277227723 | 0.7289719626168225 |
|  EditedNearestNeighbours  | 0.8725490196078431 | 0.8392857142857143 | 0.9215686274509803 | 0.8785046728971961 |
|          AllKNN           | 0.8404255319148937 | 0.8823529411764706 | 0.8333333333333334 | 0.8571428571428571 |
| InstanceHardnessThreshold | 0.8440366972477065 | 0.7884615384615384 | 0.8723404255319149 | 0.8282828282828283 |
|         NearMiss          | 0.6944444444444444 |       0.625        | 0.8163265306122449 | 0.7079646017699115 |
| NeighbourhoodCleaningRule | 0.7980769230769231 | 0.7538461538461538 | 0.9074074074074074 | 0.8235294117647058 |
|     OneSidedSelection     | 0.782608695652174  | 0.7297297297297297 | 0.574468085106383  | 0.6428571428571429 |
|    RandomUnderSampler     | 0.7314814814814815 | 0.6666666666666666 | 0.8163265306122449 | 0.7339449541284403 |
|        TomekLinks         | 0.7552447552447552 | 0.6382978723404256 |       0.625        | 0.631578947368421  |
+---------------------------+--------------------+--------------------+--------------------+--------------------+
```

Image 7.5

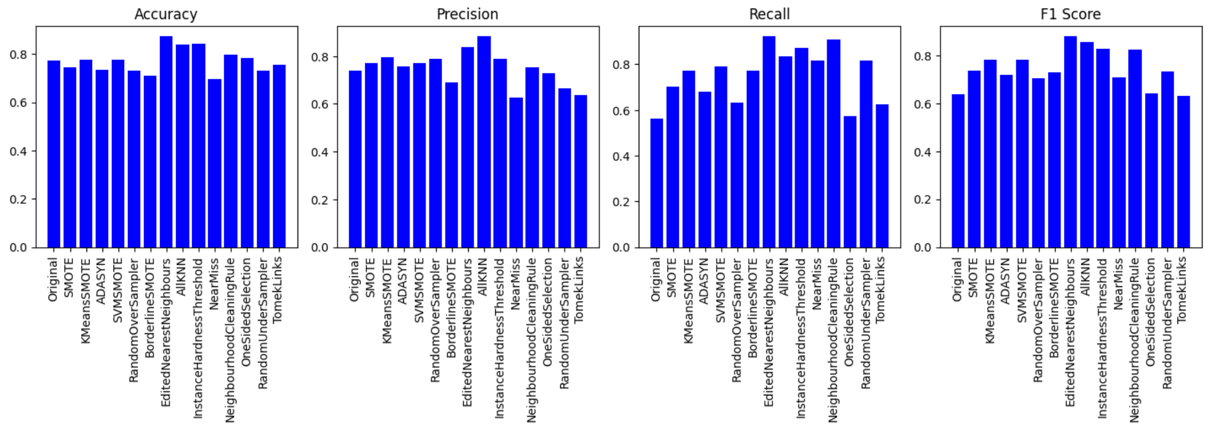


Image 7.6

As can be seen, the dataset balancing method with the highest machine learning metric values in the architecture I designed is EditedNearestNeighbours.

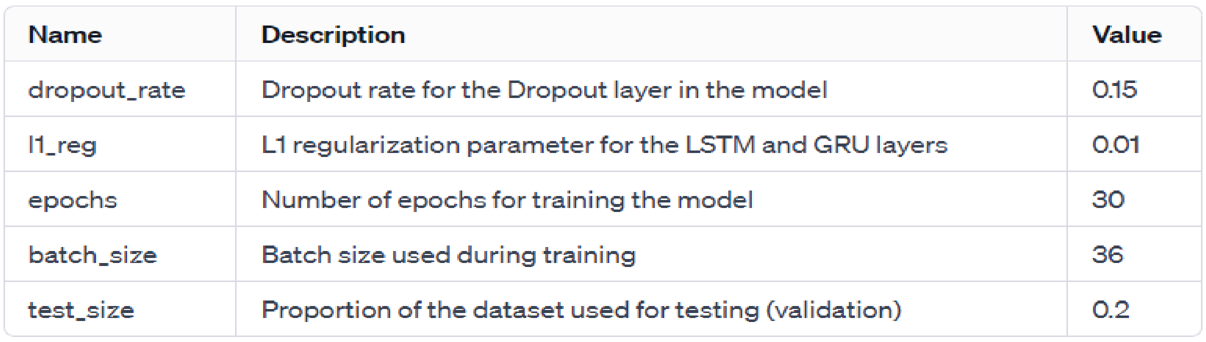
3. CNN + LSTM + GRU

Model Architecture:

- The neural network model includes Conv1D, MaxPooling1D, LSTM, GRU, Dropout, Flatten, Dense layers.
- The input data is reshaped to fit the Conv1D layer. Conv1D layer expects 3D input: (samples, time steps, features).

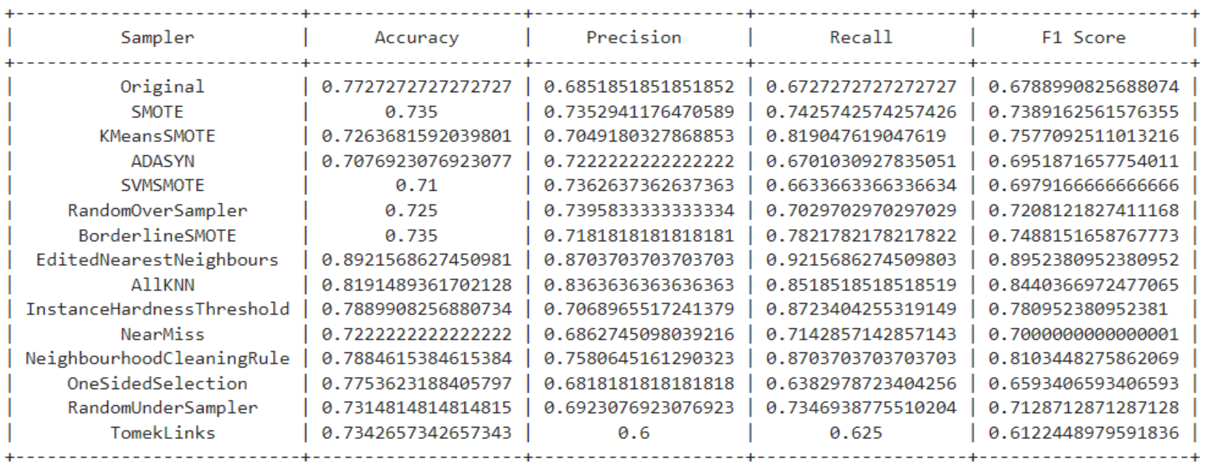
The model includes the following layers:

- Conv1D layer: 32 filters, 3 kernel sizes and ReLU activation.
- MaxPooling1D layer: A pooling size of 2.
- LSTM layer: 50 units, L1 regularization term and return_sequences set to True.
- GRU layer: 50 units, L1 regularization term and return_sequences set to True.
- Dropout tier: With a certain dropout rate.
- Flatten layer: To convert the output into a one-dimensional array.
- Dense layer: 32 units and ReLU activation.
- Output layer: 1 unit and sigmoid activation function for binary classification.



Name	Description	Value
dropout_rate	Dropout rate for the Dropout layer in the model	0.15
l1_reg	L1 regularization parameter for the LSTM and GRU layers	0.01
epochs	Number of epochs for training the model	30
batch_size	Batch size used during training	36
test_size	Proportion of the dataset used for testing (validation)	0.2

Image 7.7



```
+---------------------------+--------------------+--------------------+--------------------+--------------------+
|          Sampler          |      Accuracy      |     Precision      |       Recall       |      F1 Score      |
+---------------------------+--------------------+--------------------+--------------------+--------------------+
|         Original          | 0.7727272727272727 | 0.6851851851851852 | 0.6727272727272727 | 0.6788990825688074 |
|           SMOTE           |       0.735        | 0.7352941176470589 | 0.7425742574257426 | 0.7389162561576355 |
|        KMeansSMOTE        | 0.7263681592039801 | 0.7049180327868853 | 0.819047619047619  | 0.7577092511013216 |
|          ADASYN           | 0.7076923076923077 | 0.7222222222222222 | 0.6701030927835051 | 0.6951871657754011 |
|         SVMSMOTE          |        0.71        | 0.7362637362637363 | 0.6633663366336634 | 0.6979166666666666 |
|     RandomOverSampler     |       0.725        | 0.7395833333333334 | 0.7029702970297029 | 0.7208121827411168 |
|      BorderlineSMOTE      |       0.735        | 0.7181818181818181 | 0.7821782178217822 | 0.7488151658767773 |
|  EditedNearestNeighbours  | 0.8921568627450981 | 0.8703703703703703 | 0.9215686274509803 | 0.8952380952380952 |
|          AllKNN           | 0.8191489361702128 | 0.8363636363636363 | 0.8518518518518519 | 0.8440366972477065 |
| InstanceHardnessThreshold | 0.7889908256880734 | 0.7068965517241379 | 0.8723404255319149 | 0.780952380952381  |
|         NearMiss          | 0.7222222222222222 | 0.6862745098039216 | 0.7142857142857143 | 0.7000000000000001 |
| NeighbourhoodCleaningRule | 0.7884615384615384 | 0.7580645161290323 | 0.8703703703703703 | 0.8103448275862069 |
|     OneSidedSelection     | 0.7753623188405797 | 0.6818181818181818 | 0.6382978723404256 | 0.6593406593406593 |
|    RandomUnderSampler     | 0.7314814814814815 | 0.6923076923076923 | 0.7346938775510204 | 0.7128712871287128 |
|        TomekLinks         | 0.7342657342657343 |        0.6         |       0.625        | 0.6122448979591836 |
+---------------------------+--------------------+--------------------+--------------------+--------------------+
```

Image 7.8

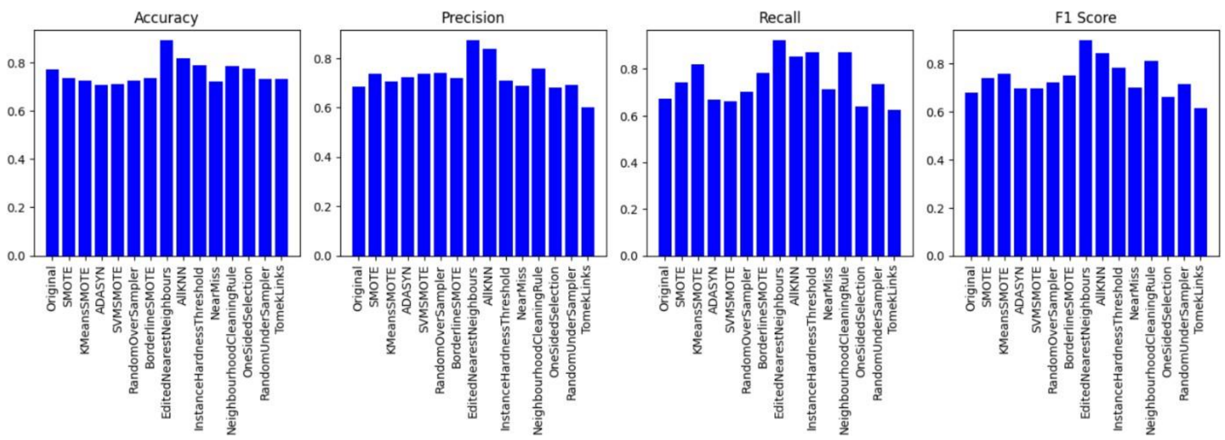
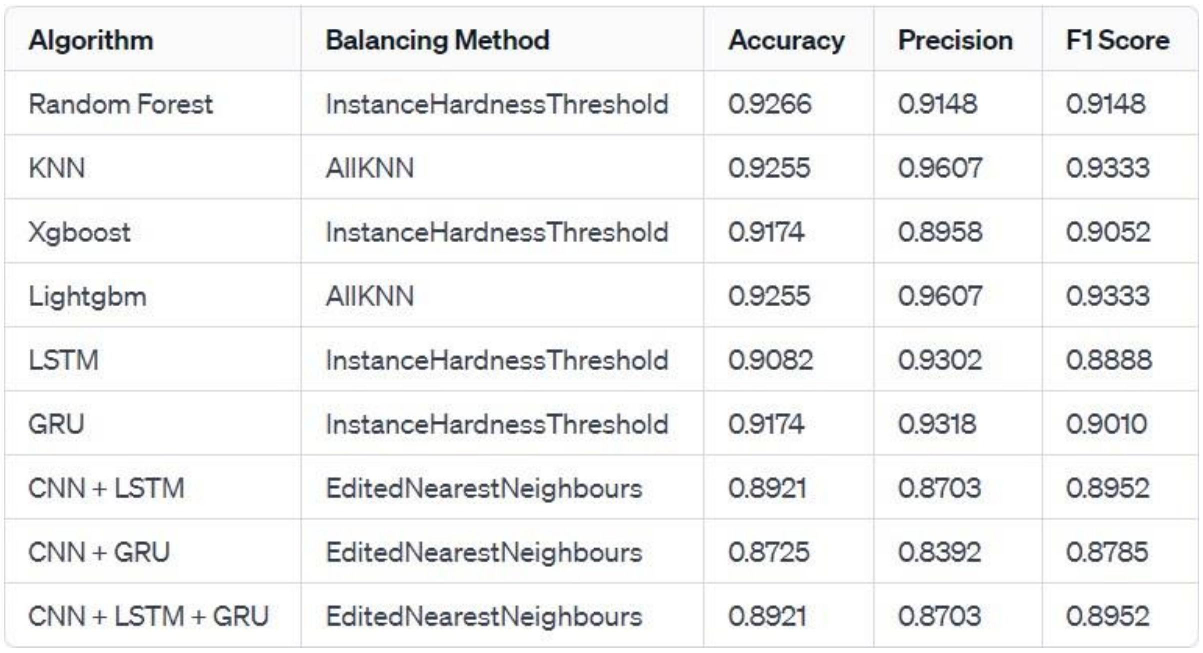


Image 7.9

As can be seen, the dataset balancing method with the highest machine learning metric values in the architecture I designed is EditedNearestNeighbours.

8. Comparing All Algorithms and Recommending Model



Algorithm	Balancing Method	Accuracy	Precision	F1 Score
Random Forest	InstanceHardnessThreshold	0.9266	0.9148	0.9148
KNN	AllKNN	0.9255	0.9607	0.9333
Xgboost	InstanceHardnessThreshold	0.9174	0.8958	0.9052
Lightgbm	AllKNN	0.9255	0.9607	0.9333
LSTM	InstanceHardnessThreshold	0.9082	0.9302	0.8888
GRU	InstanceHardnessThreshold	0.9174	0.9318	0.9010
CNN + LSTM	EditedNearestNeighbours	0.8921	0.8703	0.8952
CNN + GRU	EditedNearestNeighbours	0.8725	0.8392	0.8785
CNN + LSTM + GRU	EditedNearestNeighbours	0.8921	0.8703	0.8952

Image 8.1

Table 8.1 shows the balancing methods and metric values that give the highest machine learning algorithms in each algorithm I applied.

Machine Learning metrics are critical measurements used to evaluate the performance of the model. Confusion matrix, accuracy and loss values, which are among these metrics, are important tools to evaluate the classification ability of the model and the training process.

The confusion matrix is a table that shows how correctly or incorrectly the model classifies betIen classes. This matrix allows us to analyze the performance of the model in detail, especially in multi-class problems.

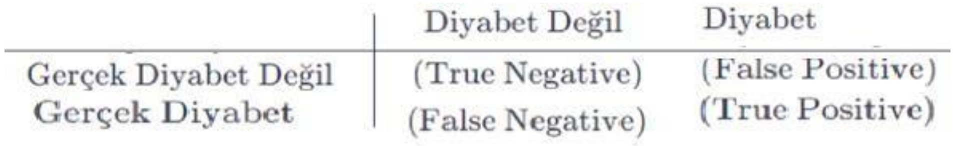
Accuracy refers to the ratio of correctly classified samples to the total number of samples. This metric is used as a measure of overall performance. Loss is a value that measures how far the model's predictions are from the actual values. LoIr loss values indicate that the model performs better.

Graphs showing the change of these metrics over time help us understand how the model improves as it is trained. For example, the accuracy over training graph shows how the model's accuracy value changes during the training process, while the loss over training graph shows how the model's loss value decreases or increases.

The use of these metrics plays a critical role in the process of assessing the reliability of the model and optimizing its performance. To put it from an academic perspective, these metrics are basic tools for making a healthy performance analysis during the training and evaluation phases of the model.

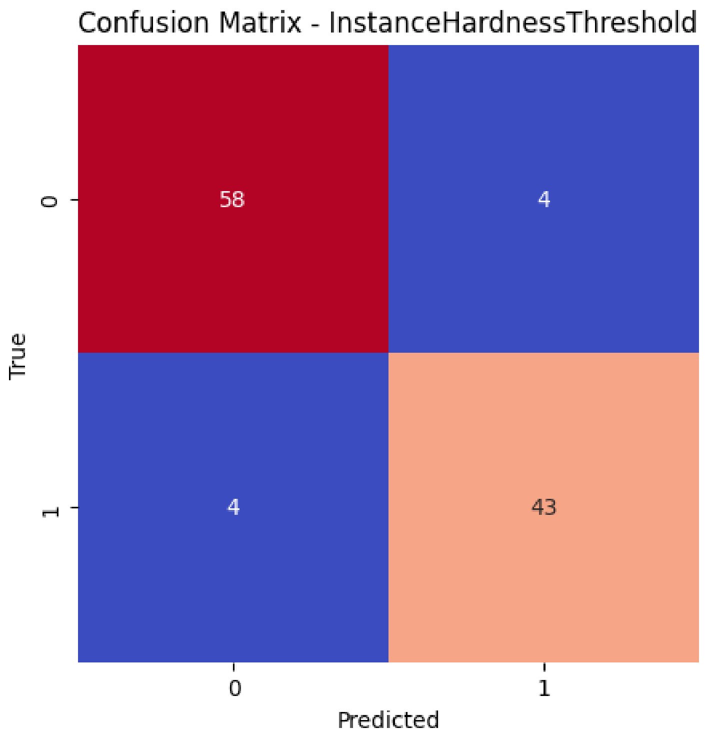
If I give general information about the confusion matrix;

- True Positive (TP): Accurately identifying true diabetics.
- True Negative (TN): Accurately detecting those who do not actually have diabetes.
- False Positive (FP): Incorrectly identifying people who do not actually have diabetes as having diabetes.
- False Negative (FN): People who actually have diabetes are incorrectly identified as not having diabetes.

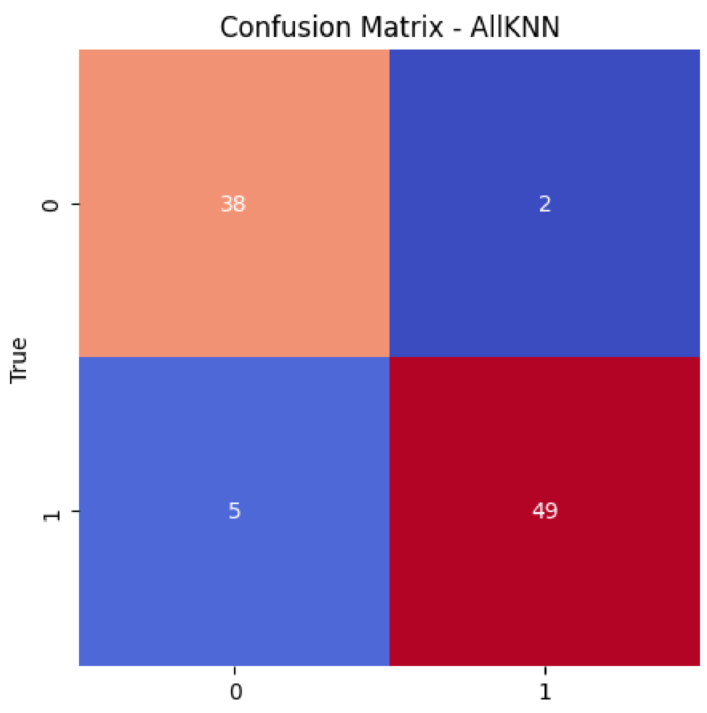


	Diyabet Değil	Diyabet
Gerçek Diyabet Değil	(True Negative)	(False Positive)
Gerçek Diyabet	(False Negative)	(True Positive)

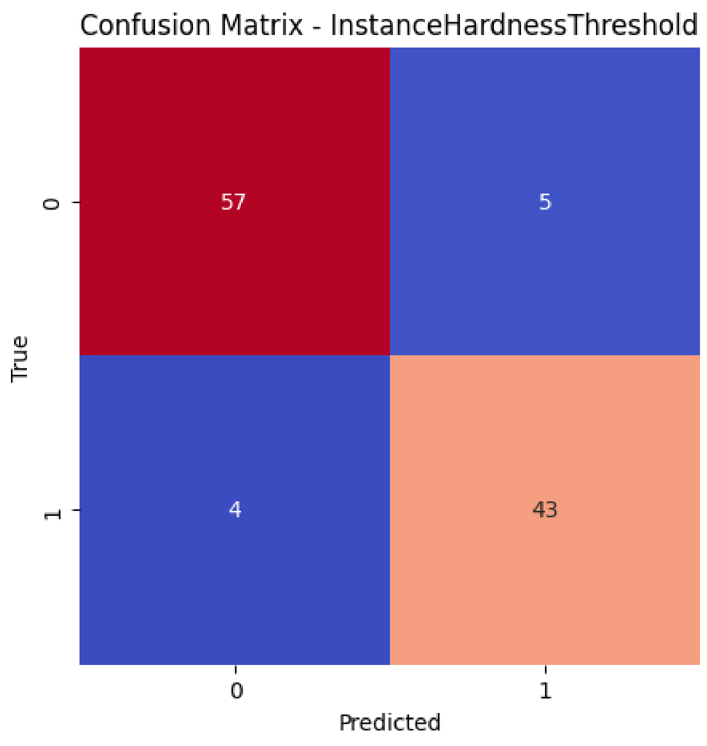
Image 8.2



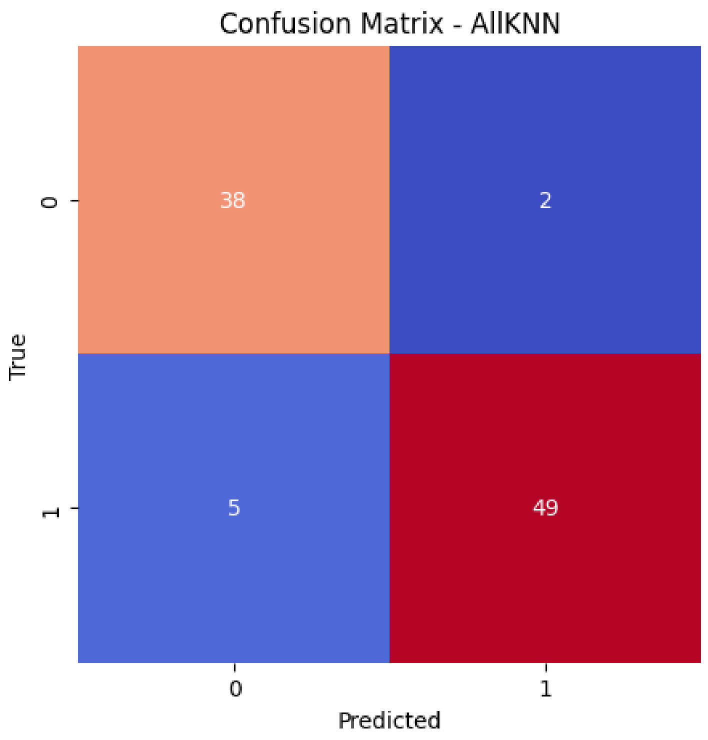
The Best in Random Forest



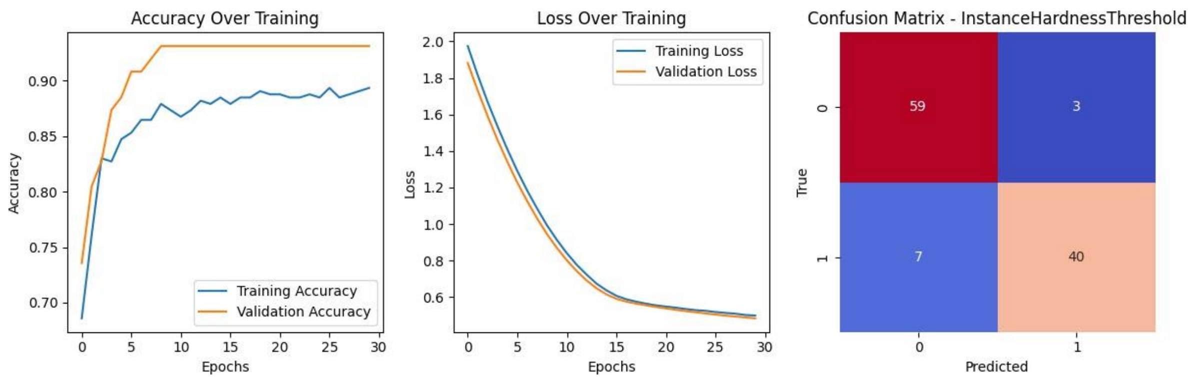
The Best in KNN



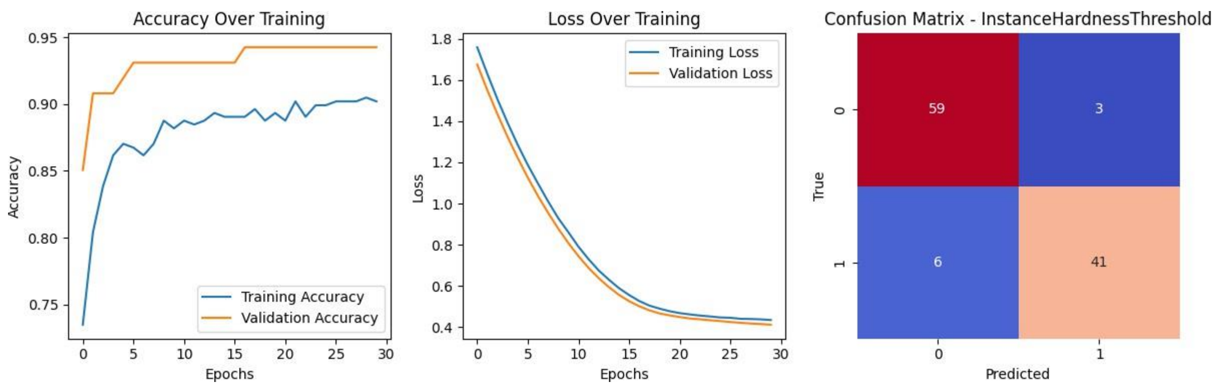
The Best in Xgboost



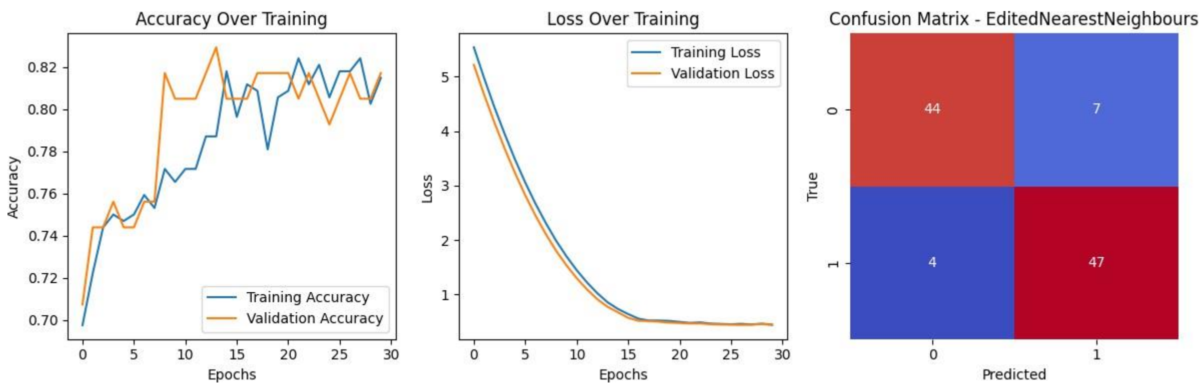
The Best in Lightgbm



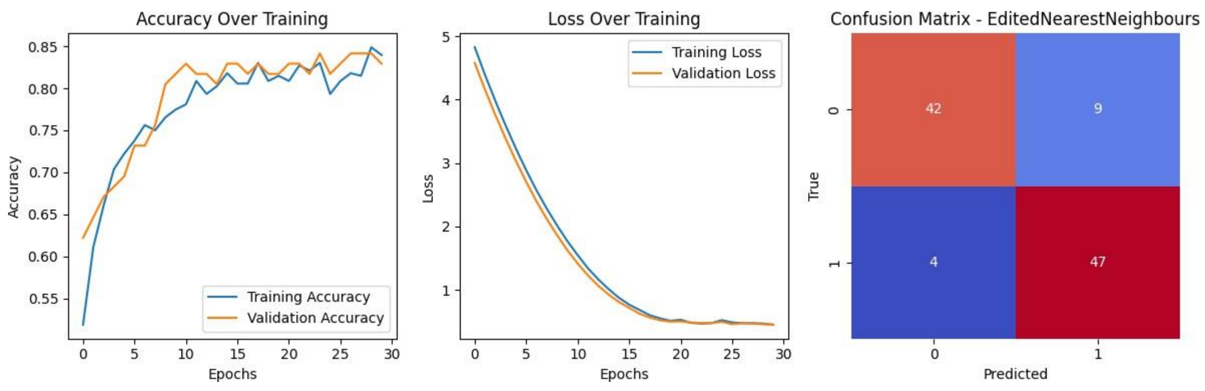
The Best in LSTM



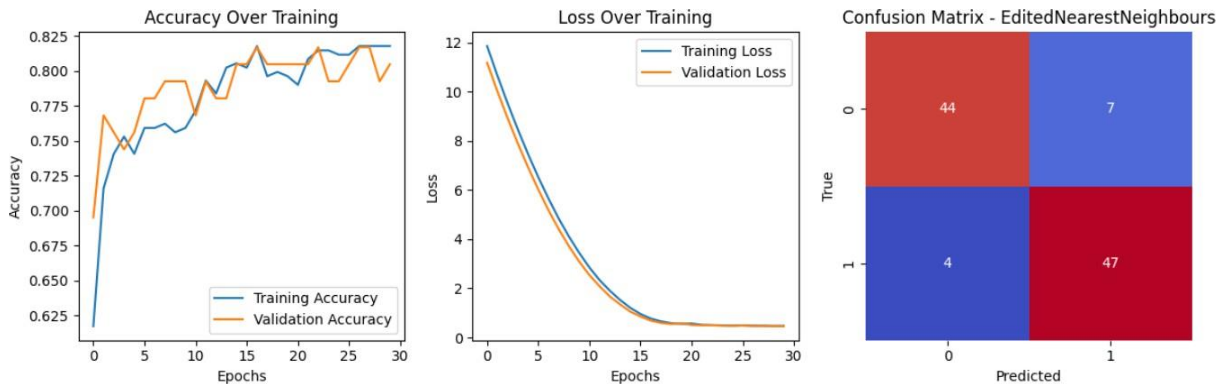
The Best in GRU



The Best in CNN+LSTM

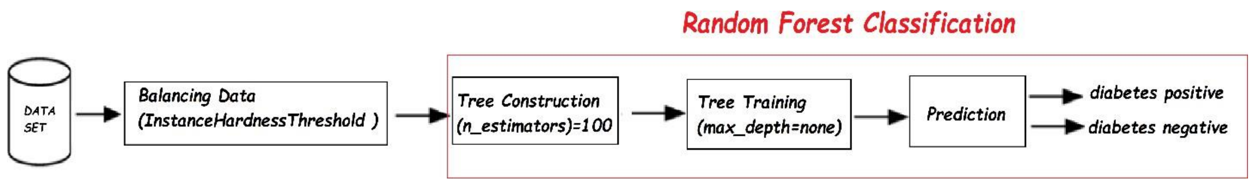


The Best in CNN+GRU



The Best in CNN+LSTM+GRU

When both the numerical machine learning metrics and the images above are examined, no serious difference is observed betlen the obtained algorithms. Due to this situation, I hesitate to say that it is more advantageous to use exactly this algorithm. Holver, despite this situation, I can say that the metric results obtained in Random Forest are quite efficient in diagnosing the disease. For this reason, the algorithm I recommend in diagnosing diabetes is to apply Random Forest after applying the InstanceThreshold balancing method to the data in this project.


Random Forest Classification
DATA SET
Balancing Data (InstanceHardnessThreshold)
Tree Construction (n_estimators)=100
Tree Training (max_depth=none)
Prediction
diabetes positive
diabetes negative

Recommended Algorithm For Detecting Diabetes

9. Resullts

This study aims to develop a predictive model for measuring diabetes. A significant portion of the global population suffers from diabetes, making it a crucial health concern. The goal of this research is to create an effective prediction model for diabetes. The performance of the model is evaluated on the Pima Indian Diabetes dataset.

Various balancing methods are applied to the dataset, and several algorithms are tested. In this context, the InstanceThreshold balancing method is applied to the dataset, folloId by the implementation of the Random Forest Classification algorithm. The balanced dataset, obtained by applying InstanceThreshold, is split into an 80% training set and a 20% test set, resulting in a 92.66% accuracy.

When compared to state-of-the-art models, the developed model demonstrates effective performance in predicting diabetes. Future studies aim to assist healthcare professionals in the early diagnosis of diabetes and plan for the development of more advanced applications.

References

https://tr.wikipedia.org/wiki/Diyabet

https://dergipark.org.tr/tr/pub/gazimmfd/issue/72928/880750

https://dergipark.org.tr/en/download/article-file/1371423

https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/