Machine Learning Project

Participants: Rafi Dinary 300241023
Reut Kishon 206966517

Github link: https://github.com/ReutKishon/Project_machine_learning.git

Dataset description:

The dataset contains medical information about patients, like gender, age, various diseases, and smoking status. Each row in the data provides relevant attributes about the patient.

Dataset source: https://www.kaggle.com/fedesoriano/stroke-prediction-dataset

Our questions:

1) predict whether a patient is likely to get a **stroke** based on the input parameters.
2) predict **blood glucose leve**l a patient is likely to have based on the input parameters.

3) predict whether a patient is likely to have a **hypertension** based on the input parameters
4) predict whether a patient is likely to have a **heart disease** based on the input parameters

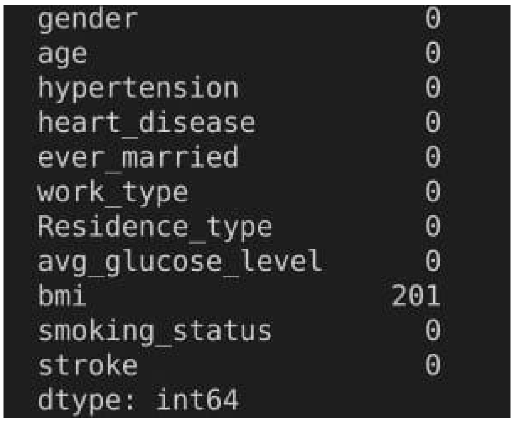
Data Preprocessing

Data preprocessing is one of the most important things, and it is one of the common factors of success of a model.

There are 4 main important steps for the preprocessing of data:

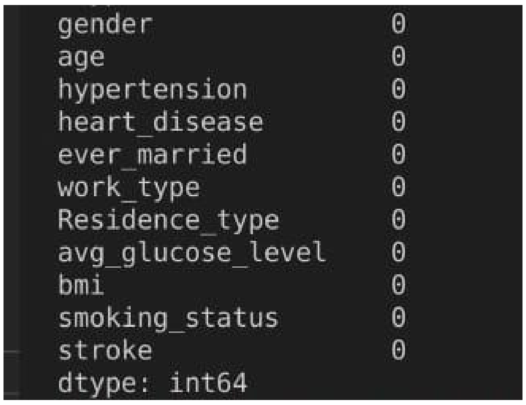
1. Splitting of the data set in Training and Validation sets:
   using *scikit-learn*, which has a built-in function *train_test_split.*
2. Taking care of Missing values:
   When the data set is full of NaNs and garbage values, then surely our model

will perform garbage too. So taking care of such missing values is important. In our case, some of the values in the ‘bmi’ column are missing (N/A). you can see it using: ***df.isna().sum()*** command:


```
gender                0
age                   0
hypertension          0
heart_disease         0
ever_married          0
work_type             0
Residence_type        0
avg_glucose_level     0
bmi                 201
smoking_status        0
stroke                0
dtype: int64
```

We complete the missing values of the bmi column with KNNImputer:
Each sample’s missing values are imputed using the mean value from n_neighbors nearest neighbors found in the training set. Two samples are close if the features that neither is missing are close.

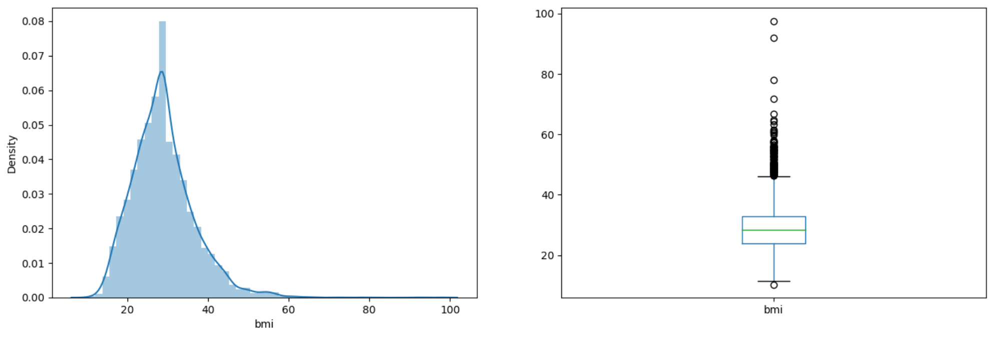
After we have filled all the missing values:


```
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                  0
smoking_status       0
stroke               0
dtype: int64
```

3. <u>Taking care of Categorical Features:</u>
   We take care of categorical features by converting them to integers using Label Encoding. In Label Encoder, we can convert the Categorical values into numerical labels. In our data frame we have five categorical columns: gender, ever_married , work_type , Residence_type , smoking_status.
   Creating a temp dataframe that contains the encoded categorical fields.

(not_num_cols). Then, we copy the temp dataframe's columns to the columns of the original data frame respectively.

4. Treating outliers: Since the BMI column has some outliers, as shown in the figure below, we had to clean it. We assign to each outlier the value of the 99.7 percentile. We tried, but did not find other columns that can be 'improved' by removing outliers.



5. Normalization of data set: Since some of the algorithms we used require that the data will be normalized to the same scale, we used sklearn MinMaxScaler to scale all the values in the data frame to be in the range [0,1].
6. selecting features: since some of the features are less usable for learning, we dropped the feature columns if their weight is less than 0.005. We use the chi-square function score for classification and the f_regression function for scoring the columns for regression.

techniques we use:

1. k-Nearest-Neighbor
2. Naive Bayes
3. Random Forest
4. decision tree algorithm

results:

In each algorithm we split the data into a 75% training set and 25% test set.

k-Nearest-Neighbor:

We wanted to choose the best hyperparameter k which will give the best result.

But there are no predefined statistical methods to find the most favorable value of k.

We split the data and choose k according to the method we found:

Starting computing the algorithm from k=1 , k=3, .. to k = the square root of the training set ($\sqrt{5111} = 71$).

The reason is that more classifiers increases computation time. This complies with what the pilot study suggests, since using this threshold was based on benefit cost, the highest accuracy with the lowest computation time.

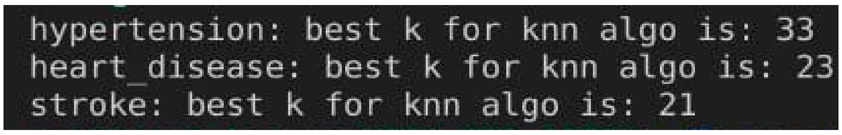
The proposed method uses the odd numbers for the k parameter for two reasons:

1) to increase the speed of the algorithm by avoiding the even classifiers.

2) to avoid the chance of two different classes having the same number of votes.

then choosing the K value that has a minimum error rate.

source - https://arxiv.org/pdf/1409.0919.pdf



```
hypertension: best k for knn algo is: 33
heart_disease: best k for knn algo is: 23
stroke: best k for knn algo is: 21
```

decision tree algorithm:

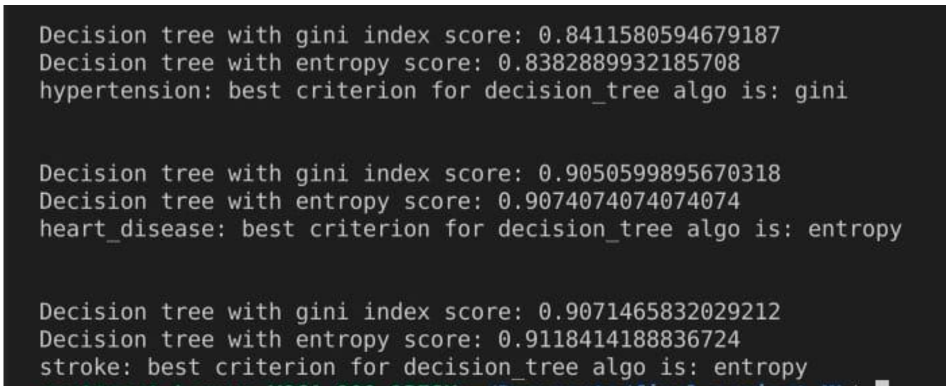
Gini index and entropy are the criteria for calculating information gain. Decision tree algorithms use information gain to split a node.

Both gini and entropy are measures of impurity of a node. A node having multiple classes is impure whereas a node having only one class is pure.

$$Gini = 1 - \sum_{i=1}^{n} p^2(c_i)$$

$$Entropy = \sum_{i=1}^{n} -p(c_i)log_2(p(c_i))$$

where $p(c_i)$ is the probability/percentage of class c_i in a node.

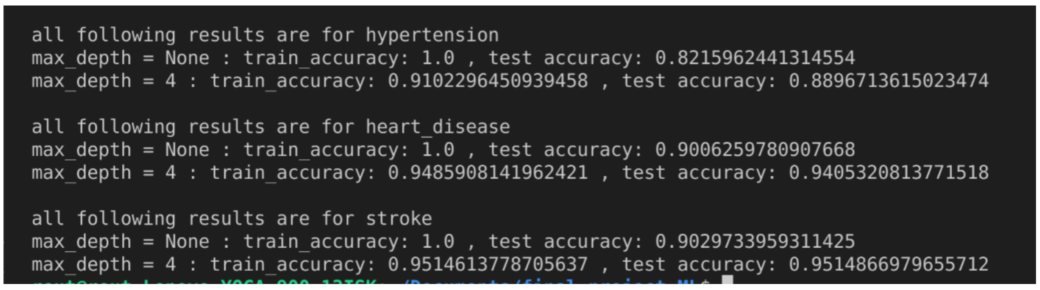


```
Decision tree with gini index score: 0.8411580594679187
Decision tree with entropy score: 0.8382889932185708
hypertension: best criterion for decision_tree algo is: gini


Decision tree with gini index score: 0.9050599895670318
Decision tree with entropy score: 0.9074074074074074
heart_disease: best criterion for decision_tree algo is: entropy


Decision tree with gini index score: 0.9071465832029212
Decision tree with entropy score: 0.9118414188836724
stroke: best criterion for decision_tree algo is: entropy
```

As we can see, there is not much performance difference when using gini index compared to entropy as a splitting criterion. Therefore any one of gini or entropy can be used as a splitting criterion.

max_depth: In general, the deeper we allow our tree to grow, the more complex our model will become because we will have more splits and it captures more information about the data and this is one of the main causes of overfitting in decision trees - because the model will fit perfectly for the training data and will not be able to generalize well on test set.
So, we checked if our model is overfitting with max_depth = None vs reducing the number for max_depth to 4.

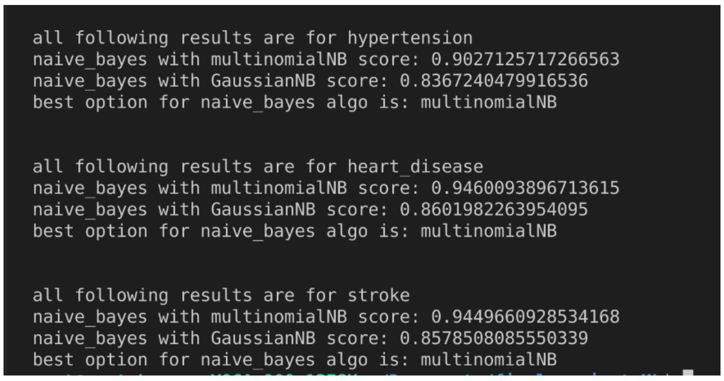
<u>results:</u>



```
all following results are for hypertension
max_depth = None : train_accuracy: 1.0 , test accuracy: 0.8215962441314554
max_depth = 4 : train_accuracy: 0.9102296450939458 , test accuracy: 0.8896713615023474

all following results are for heart_disease
max_depth = None : train_accuracy: 1.0 , test accuracy: 0.9006259780907668
max_depth = 4 : train_accuracy: 0.9485908141962421 , test accuracy: 0.9405320813771518

all following results are for stroke
max_depth = None : train_accuracy: 1.0 , test accuracy: 0.9029733959311425
max_depth = 4 : train_accuracy: 0.9514613778705637 , test accuracy: 0.9514866979655712
```

In max_depth = None : the model performs better on the training set than on the test set, it means that the model is likely overfitting.
In contrast to max_depth = 4 : the model performs similarly in both training set and test set.

Naive Bayes:
The results show that the Multinomial Naive Bayes has much better accuracy when compared with the Gaussian technique given the same dataset and parameters.
Multinomial average: 0.93 , Gaussian average: 0.83
We think the reason is because multinomial Naive Bayes classifiers are suitable for classification with discrete features, in our data the features are discrete (e.g: gender, smoking status, bmi..).



```
all following results are for hypertension
naive_bayes with multinomialNB score: 0.9027125717266563
naive_bayes with GaussianNB score: 0.8367240479916536
best option for naive_bayes algo is: multinomialNB


all following results are for heart_disease
naive_bayes with multinomialNB score: 0.9460093896713615
naive_bayes with GaussianNB score: 0.8601982263954095
best option for naive_bayes algo is: multinomialNB


all following results are for stroke
naive_bayes with multinomialNB score: 0.9449660928534168
naive_bayes with GaussianNB score: 0.8578508085550339
best option for naive_bayes algo is: multinomialNB
```

Random forest:
There is the n_estimators hyperparameter, which is just the number of trees the algorithm builds before taking the maximum voting or taking the averages of predictions. In general, a higher number of trees increases the performance and makes the predictions more stable, but it also slows down the computation

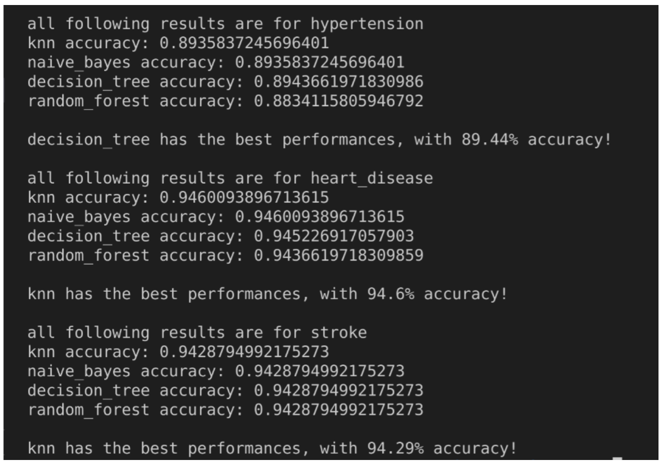
We have tried to increase the n_estimators,but the accuracy has not changed at all and it slowed down the computation. Hence we use 100 trees in the algorithm as default.

Comparison of all algorithms:
We've made a comparison of all the algorithms we mentioned above:
The algorithms are fit and evaluated on the same subsets of the dataset.

In every question: it chooses the best k using the 'check_best_k_for_knn' function for the knn algorithm.

In the decision tree algorithm,we used the default criterion - gini. (as we already explained : there is not a big difference between gini and entropy accuracy.) and used with max_depth = 4.

In Naive Bayes: we used the multinomial Naive Bayes classifier that suits our data.



```
all following results are for hypertension
knn accuracy: 0.8935837245696401
naive_bayes accuracy: 0.8935837245696401
decision_tree accuracy: 0.8943661971830986
random_forest accuracy: 0.8834115805946792

decision_tree has the best performances, with 89.44% accuracy!

all following results are for heart_disease
knn accuracy: 0.9460093896713615
naive_bayes accuracy: 0.9460093896713615
decision_tree accuracy: 0.945226917057903
random_forest accuracy: 0.9436619718309859

knn has the best performances, with 94.6% accuracy!

all following results are for stroke
knn accuracy: 0.9428794992175273
naive_bayes accuracy: 0.9428794992175273
decision_tree accuracy: 0.9428794992175273
random_forest accuracy: 0.9428794992175273

knn has the best performances, with 94.29% accuracy!
```

As we can see, all of the algorithms have similar accuracy.

<u>Predicting glucose level</u>

We tried to use the same data set for predicting glucose level.

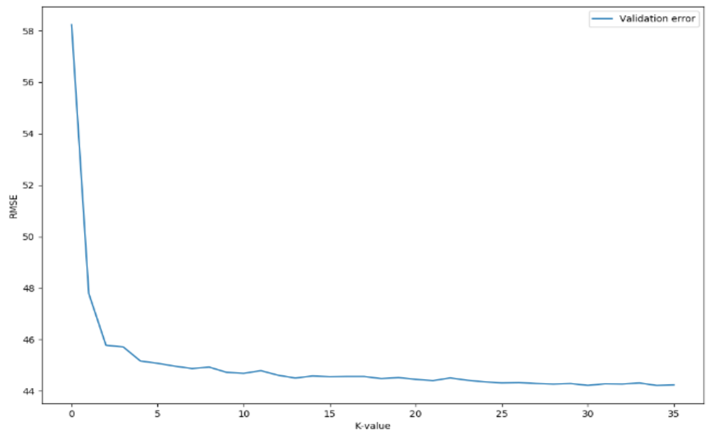
The results were unsustainable, maybe because of the small number of rows.

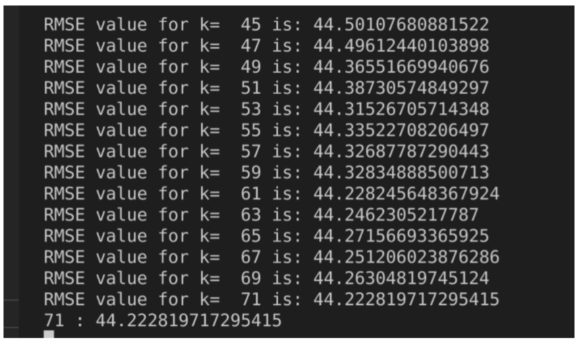
<u>Knn Regressor:</u>

We wanted to choose the best hyperparameter k which will give the best result.

In a regression problem we can measure the error rate of the algorithm with each K using Root Mean Square Error (RMSE).

Root Mean Square Error (RMSE) is the standard deviation of the prediction errors. RMSE is a measure of how spread out these prediction errors are. In other words, it tells you how concentrated the data is around the line of best fit.



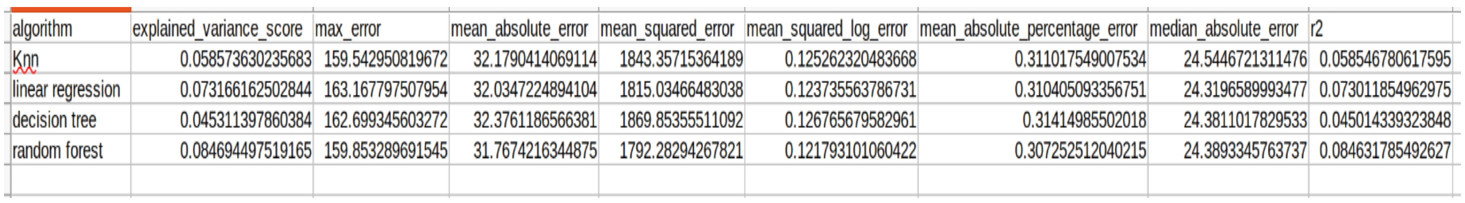


```
RMSE value for k=  45 is: 44.50107680881522
RMSE value for k=  47 is: 44.49612440103898
RMSE value for k=  49 is: 44.36551669940676
RMSE value for k=  51 is: 44.38730574849297
RMSE value for k=  53 is: 44.31526705714348
RMSE value for k=  55 is: 44.33522708206497
RMSE value for k=  57 is: 44.32687787290443
RMSE value for k=  59 is: 44.32834888500713
RMSE value for k=  61 is: 44.228245648367924
RMSE value for k=  63 is: 44.2462305217787
RMSE value for k=  65 is: 44.27156693365925
RMSE value for k=  67 is: 44.251206023876286
RMSE value for k=  69 is: 44.26304819745124
RMSE value for k=  71 is: 44.222819717295415
71 : 44.222819717295415
```

As we can see, when we take k=1, we get a very high RMSE value. The RMSE value decreases as we increase the k value. At k= 71, the RMSE is approximately 44.22, and We can safely say that k=71 will give us the best result in this case.

In the random forest and the decision tree algorithms we choose max_depth = 4. (as we explained above)

<u>running all the algorithms on the same training set and test set:</u>



algorithm	explained_variance_score	max_error	mean_absolute_error	mean_squared_error	mean_squared_log_error	mean_absolute_percentage_error	median_absolute_error	r2
Knn	0.058573630235683	159.542950819672	32.1790414069114	1843.35715364189	0.125262320483668	0.311017549007534	24.5446721311476	0.058546780617595
linear regression	0.073166162502844	163.167797507954	32.0347224894104	1815.03466483038	0.123735563786731	0.310405093356751	24.3196589993477	0.073011854962975
decision tree	0.045311397860384	162.699345603272	32.3761186566381	1869.85355511092	0.126765679582961	0.31414985502018	24.3811017829533	0.045014339323848
random forest	0.084694497519165	159.853289691545	31.7674216344875	1792.28294267821	0.121793101060422	0.307252512040215	24.3893345763737	0.084631785492627

Our Analyse:

- We can learn from the table that the ‘best’ algorithm in any param is the random forest algorithm, but the results are still poor.
  There is no big difference between the values.

- Generally, Random Forests produce better results, work well on large datasets.
- Linear regression is a linear model, which means it works really nicely when the data has a linear shape. But, since our data has a non-linear shape, then a linear model cannot capture the non-linear features.
- So in this case, decision tree algorithms do a better job at capturing the non-linearity in the data by dividing the space into smaller sub-spaces depending on the questions asked.

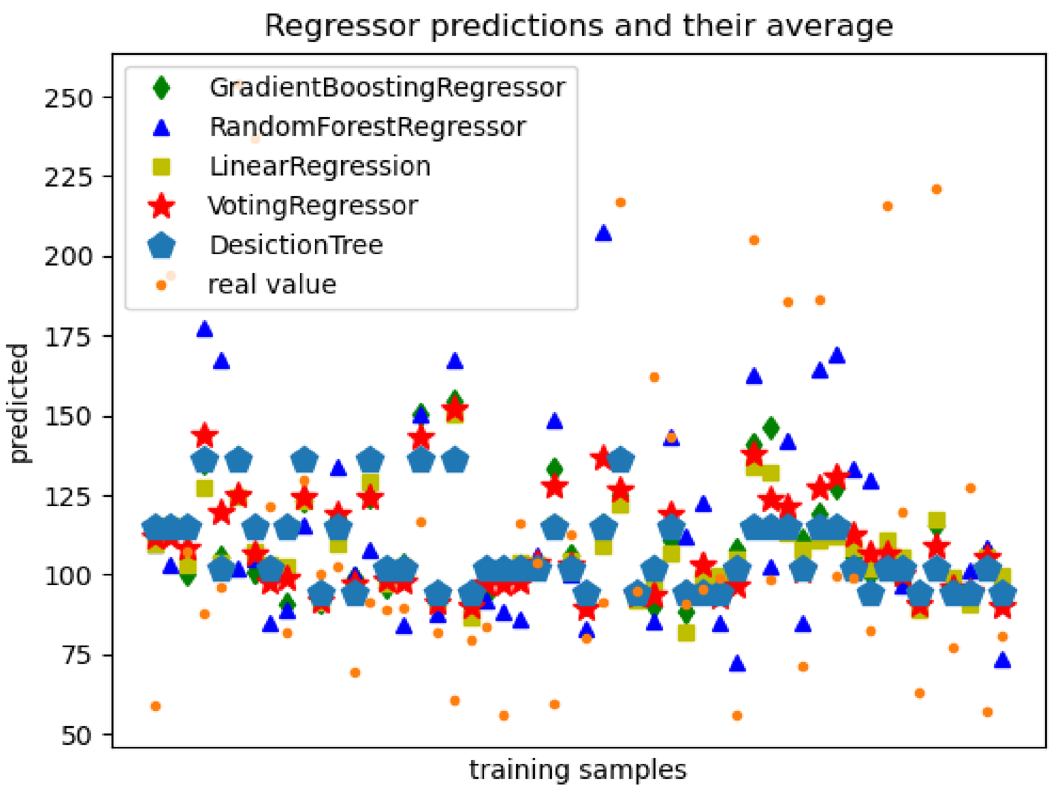
- Random forest is an ensemble of decision trees. This is to say that many trees, constructed in a certain “random” way form a Random Forest.
  Each of the trees makes its own individual prediction.
  These predictions are then averaged to produce a single result.
  The averaging makes a Random Forest better than a single Decision Tree hence improves its accuracy and reduces overfitting.

- Random forest leverages the power of multiple decision trees.
  It does *not* rely on the feature importance given by a single decision tree.

To sum up, all the algorithms predicted very poorly trying to predict the values of the average glucose value.
We can assume that more data is needed.
Also since the data set was not built for predicting glucose level but for predicting stroke the results are so poor. Also, the stroke prediction is a classification challenge,

making the data fit for this type of prediction, and not for predicting the glucose level which is a regression task.

an example to the poor results, we can see in the following figure:


Regressor predictions and their average
GradientBoostingRegressor
RandomForestRegressor
LinearRegression
VotingRegressor
DesictionTree
real value
predicted
training samples