ML Project Report

September 2022

1 Introduction

GDM (gestational diabetes mellitus) is a diabetes that occurs during pregnancy. About 17-63% of pregnant women with GDM develop type 2 diabetes 5-16 years after delivery; The recurrence rate of GDM in another pregnancy is as high as 52-69%. Approximately 3 to 8 percent of all pregnant women in the United States are diagnosed with gestational diabetes[1]. Diagnosing GDM (gestational diabetes mellitus) is a complex undertaking. The symptoms of diabetes are reflected in multiple indicators. which increases the difficulty of diagnosis and reduces accuracy. In this project, an efficient and interpretable algorithm was designed to diagnose diabetes based on multiple clinical data and physical examination indicators of patients.

The structure of this report is as follows: section 1 of this report is an introduction to the topic, section 2 shows the idea to formalize the above-mentioned task a machine learning problem, as well as the information on the data points, features and labels of this ML problem. In section 3, the method of the experiment is explicitly discussed. section 4 then presents the ML methods used along with their results, and section 5 is the conclusions.

2 Problem Formulation

Our model implements a binary classification supervised learning task to predict whether the person has diabetes through multiple features. The prediction result is whether each person has gestational diabetes, the category is expressed as an integer, and the value is 0 for false or 1 for true.

The first line of the dataset csv file is the field name, each subsequent line represents an individual, and some field names have been desensitized. The dataset contains a total of 84 feature fields, including float and int types. The first column is the individual ID number. The last column as the label column, which is the class label that needs to be predicted whether it is diseased or not. The rest columns are the clinical data and physical examination indicators of patients, which are used as features.

Except for id, SNP1, SNP2, all the features are missing in some populations. Therefore, the preprocessing is an important task before training the model.

The dataset is downloaded from Tianchi Precision Medicine Competition dataset - Artificial intelligence-assisted diabetes genetic risk prediction [2].

3 Methods

Here is the main methods of the experiment and the technical points.

3.1 Data Processing

The dataset contains 1000 samples, each sample has 84 features. Most of the features have missing values, which need to be completed with median through the method data[col] = data[col].fillna(data[col].median()) If a feature is missing too much, consider removing it or completing it with a special value. Here we completed the ‘RBP4’ and ‘childbirth time’ with 0.

3.2 Feature engineering

The project removes useless features, whose variance is less than 0.5 or low correlation with label, and only keep the first 20 features that are positively correlated and the last 15 that are negatively correlated. Taking VAR00007 as an example (Fig. 1), it can be seen that the larger the value, the higher the possibility of diabetes. The correlation is shown in the Fig. 2.

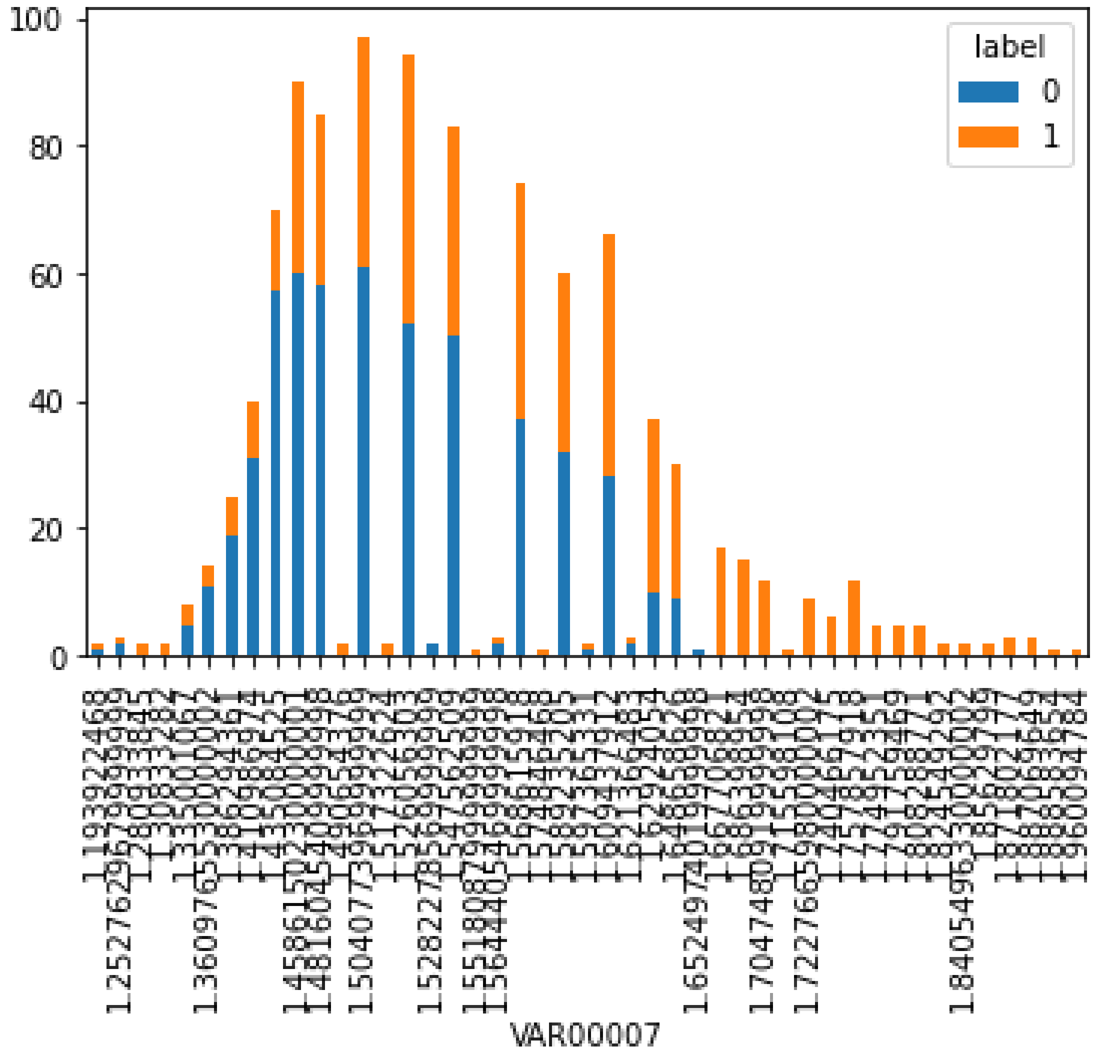


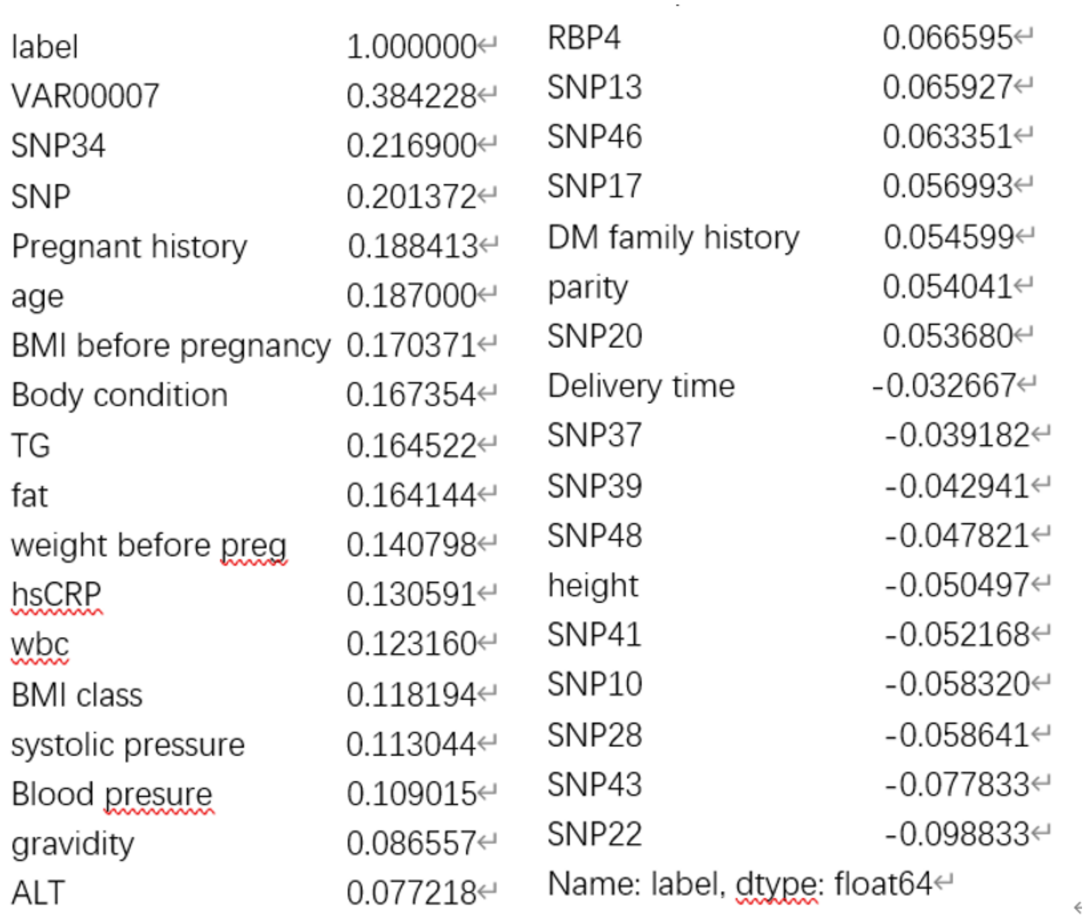
Figure 1: The relation between feature VAR00007 and the label

Here is the ablation experiment for correlation filtering. The $_corr\_matrix$ contain the correlation values of feature in descending order. $_pos\_corr$ contains top 20 features with the highest positive correlation and $_neg\_corr$ contains top 15 features with the most negative correlation. Other features with correlation around 0 is abandoned. The each feature number is adjusted from 0 to 20. The score is shown in the Table 1.

number of features	0	10	15	20
test score	0.781	0.897	0.95	0.937

Table 1: Test score with different number of features

```
corr_matrix = data.corr()
corr_matrix = corr_matrix['label'].sort_values(ascending=False)
pos_corr = list(corr_matrix[1:20].index)
neg_corr = list(corr_matrix[-15:].index)
cols = data.columns.values
for col in cols:
    if((col not in pos_corr)
            and(col not in neg_corr)
            and (col!='label')):
        data = data.drop(col, axis=1)
```



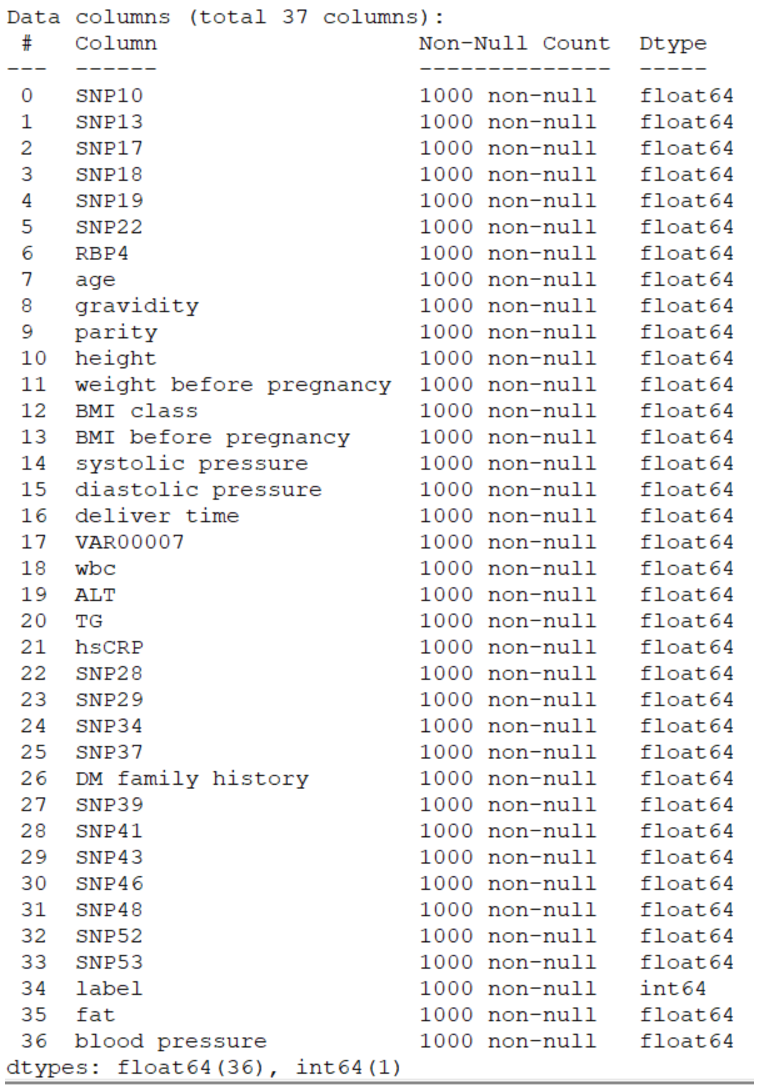
```
label                 1.000000
VAR00007              0.384228
SNP34                 0.216900
SNP                   0.201372
Pregnant history      0.188413
age                   0.187000
BMI before pregnancy  0.170371
Body condition        0.167354
TG                    0.164522
fat                   0.164144
weight before preg    0.140798
hsCRP                 0.130591
wbc                   0.123160
BMI class             0.118194
systolic pressure     0.113044
Blood presure         0.109015
gravidity             0.086557
ALT                   0.077218
RBP4                  0.066595
SNP13                 0.065927
SNP46                 0.063351
SNP17                 0.056993
DM family history     0.054599
parity                0.054041
SNP20                 0.053680
Delivery time        -0.032667
SNP37                -0.039182
SNP39                -0.042941
SNP48                -0.047821
height               -0.050497
SNP41                -0.052168
SNP10                -0.058320
SNP28                -0.058641
SNP43                -0.077833
SNP22                -0.098833
Name: label, dtype: float64
```

Figure 2: correlation

According to common medical knowledge, many important features are highly correlated, so we added some combined features to highlight their practical significance, such as pregnancy history, obesity, blood pressure, physical condition, and single nucleotide polymorphism SNPs. The data type of each feature is shown in the Fig. 3.

3.3 The construction of training and validation sets

This project uses 5-fold cross-validation to obtain average score. Because the dataset in this experiment is small for only 1000 samples, and k-fold cross validation is quite suitable for this situation. Each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times, which can significantly increase the training times and make the model more likely to converge. It generally results in



```
Data columns (total 37 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   SNP10                     1000 non-null   float64
 1   SNP13                     1000 non-null   float64
 2   SNP17                     1000 non-null   float64
 3   SNP18                     1000 non-null   float64
 4   SNP19                     1000 non-null   float64
 5   SNP22                     1000 non-null   float64
 6   RBP4                      1000 non-null   float64
 7   age                       1000 non-null   float64
 8   gravidity                 1000 non-null   float64
 9   parity                    1000 non-null   float64
 10  height                    1000 non-null   float64
 11  weight before pregnancy   1000 non-null   float64
 12  BMI class                 1000 non-null   float64
 13  BMI before pregnancy      1000 non-null   float64
 14  systolic pressure         1000 non-null   float64
 15  diastolic pressure        1000 non-null   float64
 16  deliver time              1000 non-null   float64
 17  VAR00007                  1000 non-null   float64
 18  wbc                       1000 non-null   float64
 19  ALT                       1000 non-null   float64
 20  TG                        1000 non-null   float64
 21  hsCRP                     1000 non-null   float64
 22  SNP28                     1000 non-null   float64
 23  SNP29                     1000 non-null   float64
 24  SNP34                     1000 non-null   float64
 25  SNP37                     1000 non-null   float64
 26  DM family history         1000 non-null   float64
 27  SNP39                     1000 non-null   float64
 28  SNP41                     1000 non-null   float64
 29  SNP43                     1000 non-null   float64
 30  SNP46                     1000 non-null   float64
 31  SNP48                     1000 non-null   float64
 32  SNP52                     1000 non-null   float64
 33  SNP53                     1000 non-null   float64
 34  label                     1000 non-null   int64
 35  fat                       1000 non-null   float64
 36  blood pressure            1000 non-null   float64
dtypes: float64(36), int64(1)
```

Figure 3: data type of features

a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

Specifically, the original data is split into 5 sets, each of which accounts for 20%. Each set will be taken as the testing set for once, while the remaining sets will be used to train the model.

3.4 Model selection

The experiment uses random forest model. It can handle very high dimensional (many features) data and indicate which features are more important after training is complete, which is suitable for the dataset, which have totally 84 features. A large part of the features are missing in the dataset, but random forest can still maintain the accuracy. Besides, logistic regression is easy to implement and time-efficient, which is widely applied on real-world binary classification tasks; SVM and decision tree classifier are both suitable for problems with a small dataset and high-dimensional data; Naive Bayes classifier usually shows good performance when dataset is small. Hence, the experiment do comparison among these models.
Before parameter tuning, the f1_scores of logistic regression, SVM, Naive Bayes, decision tree and random forest classifer are separately 0.667, 0.524, 0.633, 0.601, 0.674. Random forest achieves the best performance among these models. So we choose that as the final model.

3.5 Loss function

In classification tasks, we want to maximize both precision and recall. They reflect the predictive performance of a model in different ways:
Precision: Of all positive predictions, how many are really positive?

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

Recall: Of all positive samples, how many are correctly predicted to be positive?

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

Specifically, in our project, it would be most desirable that no healthy person get wrongly diagnosed while no patients are left out. In practice, however, it is not possible to maximize both of them at the same time because of the trade-off between precision and recall. Besides, there could be some problems if we choose only one of them to be the loss function: on the one hand, when precision is chosen, the model will get a perfect score by diagnosing everyone to get diabetes; on the other hand, the model tends to give negative predictions instead if recall is selected. Hence, in this project, F1 Score is chosen as the loss function, which takes both precision and recall into account.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3}$$

4 Results

Before fine tuning, the f1 score of logistic regression, SVM, Naive Bayes, decision tree, and random forest classifier are shown in Table 2.

model	Logistic regression	SVM	Naive Bayes	Decision Tree	Random forest
F1 score	0.667	0.524	0.633	0.601	0.674

Table 2: F1 score of different model before fine tuning

As mentioned, this project uses 5-fold cross validation to train and validate model. For each round, GridSearchCV is adopted to find best parameters from the set of the grid of parameters: {'n_estimators':[10,50,100, 150],'max_depth':[3,6,9,12],'criterion':['gini','entropy']}. The best parameters and F1 score of models for each round is shown in Table 3.

		model(round)				
		model(1)	model(2)	model(3)	model(4)	model(5)
parameters	criterion	gini	entropy	entropy	gini	gini
	max_depth	3	3	3	3	9
	n_estimators	150	100	50	10	50
F1 score		0.705	0.675	0.715	0.665	0.645

Table 3: Parameters and F1 score of models for each round

Model(3), which shows the best F1 score, is selected to be the final model. Besides, average F1 score of all 5 rounds is computed to reflect the average performance of random forest classifier in this project, which is generally less biased than evaluation with simple train/test split. The average F1 score after fine tunning is 0.681.

5 Conclusion

We summarize here the experimental method and the experience gained from the project.

In terms of data preprocessing, some features with too many missing data should be removed or completed with special values. In feature engineering, according to the ablation experiment, removing features with zero variance and abnormality, filtering features according to feature correlation and adding combined features can significantly improve the testing accuracy. In terms of data set division, two ways can be applied to divide the data set: use function train_test_split and use K-fold cross-validation. When the dataset is small, using K-fold cross-validation can obtain more accurate local scores.

However, the training accuracy of the model still has the space to improve. If the dataset is large enough and the amount of features is large, a relatively good subset of features may be quickly obtained in a few iterations by a method similar to genetic algorithm (GA)[3]. This type of automatic feature screening method is less difficult to implement, and usually has a better effect than manual screening, and has always been favored by the industry. This method requires sufficient computing resources, we can try to add that method to feature engineering pro in the future on servers with more powerful hardware in the future.

6 Reference

[1] John Hopkins Medicine, Gestational Diabetes Mellitus (GDM), https://www.hopkinsmedicine.org/health/conditions-and-diseases/diabetes/gestational-diabetes
[2] Tianchi Precision Medicine Competition - Artificial Intelligence-Assisted Diabetes Genetic Risk Prediction, https://tianchi.aliyun.com/competition/entrance/231638/information
[3] Mathworks, Genetic Algorithm Terminology, https://se.mathworks.com/help/gads/some-genetic-algorithm-terminology.html

Appendix

October 9, 2022

appendix:
Code and obtained result

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```
data = pd.read_csv(r'f_train.csv',encoding = 'gb2312')
```

[3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 85 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   id                                  1000 non-null   int64
 1   SNP1                                1000 non-null   int64
 2   SNP2                                1000 non-null   int64
 3   SNP3                                948 non-null    float64
 4   SNP4                                989 non-null    float64
 5   SNP5                                935 non-null    float64
 6   SNP6                                987 non-null    float64
 7   SNP7                                997 non-null    float64
 8   SNP8                                995 non-null    float64
 9   SNP9                                995 non-null    float64
 10  SNP10                               995 non-null    float64
 11  SNP11                               983 non-null    float64
 12  SNP12                               985 non-null    float64
 13  SNP13                               978 non-null    float64
 14  SNP14                               953 non-null    float64
 15  SNP15                               970 non-null    float64
 16  SNP16                               977 non-null    float64
 17  SNP17                               974 non-null    float64
 18  SNP18                               970 non-null    float64
 19  SNP19                               959 non-null    float64
 20  SNP20                               910 non-null    float64
 21  SNP21                               461 non-null    float64
```

```
22  SNP22                             460 non-null    float64
23  SNP23                             462 non-null    float64
24  RBP4                              89 non-null     float64
25  age                               975 non-null    float64
26  gravidity                         802 non-null    float64
27  parity                            802 non-null    float64
28  height                            797 non-null    float64
29  weight before pregnancy           797 non-null    float64
30  BMI class                         796 non-null    float64
31  BMI before pregnancy              796 non-null    float64
32  systolic pressure                 753 non-null    float64
33  diastolic pressure                754 non-null    float64
34  deliver time                      185 non-null    float64
35  Sugar screening week of pregnancy 795 non-null    float64
36  VAR00007                          990 non-null    float64
37  wbc                               887 non-null    float64
38  ALT                               854 non-null    float64
39  AST                               746 non-null    float64
40  Cr                                845 non-null    float64
41  BUN                               844 non-null    float64
42  CHO                               959 non-null    float64
43  TG                                959 non-null    float64
44  HDLC                              955 non-null    float64
45  LDLC                              956 non-null    float64
46  ApoA1                             934 non-null    float64
47  ApoB                              934 non-null    float64
48  Lpa                               934 non-null    float64
49  hsCRP                             925 non-null    float64
50  SNP24                             933 non-null    float64
51  SNP25                             992 non-null    float64
52  SNP26                             983 non-null    float64
53  SNP27                             987 non-null    float64
54  SNP28                             985 non-null    float64
55  SNP29                             985 non-null    float64
56  SNP30                             948 non-null    float64
57  SNP31                             952 non-null    float64
58  SNP32                             950 non-null    float64
59  SNP33                             977 non-null    float64
60  SNP34                             949 non-null    float64
61  SNP35                             969 non-null    float64
62  SNP36                             970 non-null    float64
63  SNP37                             955 non-null    float64
64  SNP38                             968 non-null    float64
65  DM family history                 700 non-null    float64
66  SNP39                             954 non-null    float64
67  SNP40                             955 non-null    float64
68  SNP41                             956 non-null    float64
69  SNP42                             962 non-null    float64
```

```
 70   SNP43                                   969 non-null    float64
 71   SNP44                                   957 non-null    float64
 72   SNP45                                   968 non-null    float64
 73   SNP46                                   929 non-null    float64
 74   SNP47                                   953 non-null    float64
 75   SNP48                                   978 non-null    float64
 76   SNP49                                   983 non-null    float64
 77   SNP50                                   981 non-null    float64
 78   SNP51                                   981 non-null    float64
 79   SNP52                                   987 non-null    float64
 80   SNP53                                   956 non-null    float64
 81   SNP54                                   483 non-null    float64
 82   SNP55                                   483 non-null    float64
 83   ACEID                                   483 non-null    float64
 84   label                                   1000 non-null   int64
dtypes: float64(81), int64(4)
memory usage: 664.2 KB
```

[4]:
```
### data preprocessing
# for most features: fill the missing value with median
cols = data.columns.values
for col in cols:
    if((col!='RBP4')
      and(col!='deliver time')):
        data[col] = data[col].fillna(data[col].median())
# for features missing too much: fill missing value with zero
data['RBP4'] = data['RBP4'].fillna(0)
data['deliver time'] = data['deliver time'].fillna(0)
```

[5]:
```
#### feature engineering
data = data.drop(['id'], axis=1)
# remove features with zero std
remove = []
for column in data.columns:
    if data[column].std()==0:
        remove.append(column)
data = data.drop(remove, axis=1)
# remove abnormal features
unique = []
for column in data.columns:
    num = len(data[column].unique())
    if data[column].isnull().sum()!=0:
        num -= 1
    if num == 1:
        unique.append(column)
data = data.drop(unique, axis=1)
data.info()
```

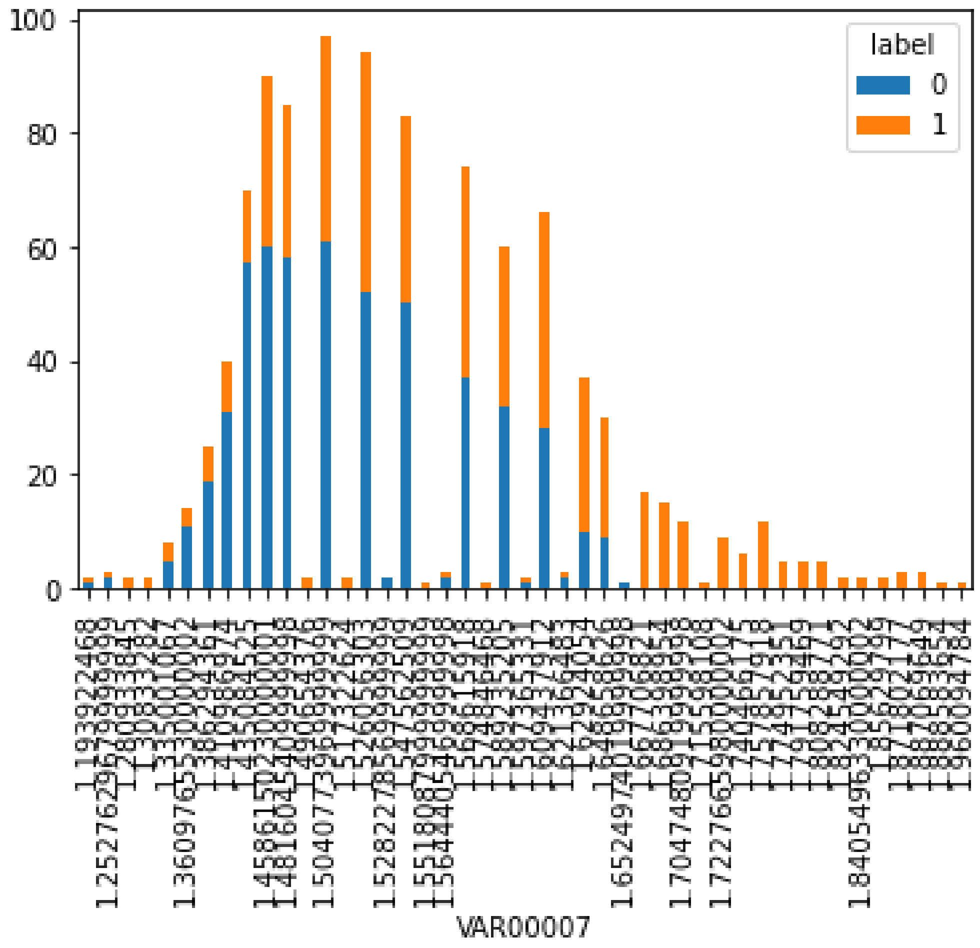
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 84 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   SNP1                              1000 non-null   int64
 1   SNP2                              1000 non-null   int64
 2   SNP3                              1000 non-null   float64
 3   SNP4                              1000 non-null   float64
 4   SNP5                              1000 non-null   float64
 5   SNP6                              1000 non-null   float64
 6   SNP7                              1000 non-null   float64
 7   SNP8                              1000 non-null   float64
 8   SNP9                              1000 non-null   float64
 9   SNP10                             1000 non-null   float64
 10  SNP11                             1000 non-null   float64
 11  SNP12                             1000 non-null   float64
 12  SNP13                             1000 non-null   float64
 13  SNP14                             1000 non-null   float64
 14  SNP15                             1000 non-null   float64
 15  SNP16                             1000 non-null   float64
 16  SNP17                             1000 non-null   float64
 17  SNP18                             1000 non-null   float64
 18  SNP19                             1000 non-null   float64
 19  SNP20                             1000 non-null   float64
 20  SNP21                             1000 non-null   float64
 21  SNP22                             1000 non-null   float64
 22  SNP23                             1000 non-null   float64
 23  RBP4                              1000 non-null   float64
 24  age                               1000 non-null   float64
 25  gravidity                         1000 non-null   float64
 26  parity                            1000 non-null   float64
 27  height                            1000 non-null   float64
 28  weight before pregnancy           1000 non-null   float64
 29  BMI class                         1000 non-null   float64
 30  BMI before pregnancy              1000 non-null   float64
 31  systolic pressure                 1000 non-null   float64
 32  diastolic pressure                1000 non-null   float64
 33  deliver time                      1000 non-null   float64
 34  Sugar screening week of pregnancy  1000 non-null   float64
 35  VAR00007                          1000 non-null   float64
 36  wbc                               1000 non-null   float64
 37  ALT                               1000 non-null   float64
 38  AST                               1000 non-null   float64
 39  Cr                                1000 non-null   float64
 40  BUN                               1000 non-null   float64
 41  CHO                               1000 non-null   float64
 42  TG                                1000 non-null   float64
 43  HDLC                                   1000 non-null   float64
 44  LDLC                                   1000 non-null   float64
 45  ApoA1                                  1000 non-null   float64
 46  ApoB                                   1000 non-null   float64
 47  Lpa                                    1000 non-null   float64
 48  hsCRP                                  1000 non-null   float64
 49  SNP24                                  1000 non-null   float64
 50  SNP25                                  1000 non-null   float64
 51  SNP26                                  1000 non-null   float64
 52  SNP27                                  1000 non-null   float64
 53  SNP28                                  1000 non-null   float64
 54  SNP29                                  1000 non-null   float64
 55  SNP30                                  1000 non-null   float64
 56  SNP31                                  1000 non-null   float64
 57  SNP32                                  1000 non-null   float64
 58  SNP33                                  1000 non-null   float64
 59  SNP34                                  1000 non-null   float64
 60  SNP35                                  1000 non-null   float64
 61  SNP36                                  1000 non-null   float64
 62  SNP37                                  1000 non-null   float64
 63  SNP38                                  1000 non-null   float64
 64  DM family history                      1000 non-null   float64
 65  SNP39                                  1000 non-null   float64
 66  SNP40                                  1000 non-null   float64
 67  SNP41                                  1000 non-null   float64
 68  SNP42                                  1000 non-null   float64
 69  SNP43                                  1000 non-null   float64
 70  SNP44                                  1000 non-null   float64
 71  SNP45                                  1000 non-null   float64
 72  SNP46                                  1000 non-null   float64
 73  SNP47                                  1000 non-null   float64
 74  SNP48                                  1000 non-null   float64
 75  SNP49                                  1000 non-null   float64
 76  SNP50                                  1000 non-null   float64
 77  SNP51                                  1000 non-null   float64
 78  SNP52                                  1000 non-null   float64
 79  SNP53                                  1000 non-null   float64
 80  SNP54                                  1000 non-null   float64
 81  SNP55                                  1000 non-null   float64
 82  ACEID                                  1000 non-null   float64
 83  label                                  1000 non-null   int64
dtypes: float64(81), int64(3)
memory usage: 656.4 KB
```

[6]:
```
# filter high correlation
pd.set_option('display.max_rows', None)
corr_matrix = data.corr()
corr_matrix = corr_matrix['label'].sort_values(ascending=False)
```

[7]:
```
pos_corr = list(corr_matrix[1:20].index)
neg_corr = list(corr_matrix[-15:].index)
cols = data.columns.values
for col in cols:
    if((col not in pos_corr) and(col not in neg_corr) and (col!='label')):
        data = data.drop(col, axis=1)
```

[8]:
```
# examples about the relation between features and label
pd.crosstab(data.VAR00007,data.label).plot.bar(stacked = True)
```

[8]:
```
<AxesSubplot:xlabel='VAR00007'>
```



[9]:
```
data['SNP34'].value_counts()
```

```
[9]: 1.0    541
     2.0    319
     3.0    140
     Name: SNP34, dtype: int64
```

```
[10]: data['SNP37'].value_counts()
```

```
[10]: 1.0    802
      2.0    150
      3.0     48
      Name: SNP37, dtype: int64
```

```
[11]: data['fat'] = data['BMI before pregnancy']+data['weight before␣
       ↪pregnancy']+data['TG']
      data['blood pressure'] = data['systolic pressure']+data['diastolic pressure']
      data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 37 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   SNP10                 1000 non-null   float64
 1   SNP13                 1000 non-null   float64
 2   SNP17                 1000 non-null   float64
 3   SNP18                 1000 non-null   float64
 4   SNP19                 1000 non-null   float64
 5   SNP22                 1000 non-null   float64
 6   RBP4                  1000 non-null   float64
 7   age                   1000 non-null   float64
 8   gravidity             1000 non-null   float64
 9   parity                1000 non-null   float64
 10  height                1000 non-null   float64
 11  weight before pregnancy  1000 non-null   float64
 12  BMI class             1000 non-null   float64
 13  BMI before pregnancy  1000 non-null   float64
 14  systolic pressure     1000 non-null   float64
 15  diastolic pressure    1000 non-null   float64
 16  deliver time          1000 non-null   float64
 17  VAR00007              1000 non-null   float64
 18  wbc                   1000 non-null   float64
 19  ALT                   1000 non-null   float64
 20  TG                    1000 non-null   float64
 21  hsCRP                 1000 non-null   float64
 22  SNP28                 1000 non-null   float64
 23  SNP29                 1000 non-null   float64
 24  SNP34                 1000 non-null   float64
 25  SNP37                     1000 non-null   float64
 26  DM family history         1000 non-null   float64
 27  SNP39                     1000 non-null   float64
 28  SNP41                     1000 non-null   float64
 29  SNP43                     1000 non-null   float64
 30  SNP46                     1000 non-null   float64
 31  SNP48                     1000 non-null   float64
 32  SNP52                     1000 non-null   float64
 33  SNP53                     1000 non-null   float64
 34  label                     1000 non-null   int64
 35  fat                       1000 non-null   float64
 36  blood pressure            1000 non-null   float64
dtypes: float64(36), int64(1)
memory usage: 289.2 KB
```

[13]:
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
# as the dataset in this experiment is small, we adapt 5-fold cross validation
# selected model: random forest
# model for comparison: logistic regression, svm, decision tree, naive bayes

# for random forest
gsrf_score_sum = 0
gsrf_best_score = 0
global gsrf, best_gsrf
# for logistic regression
lr_score_sum = 0
lr_best_score = 0
global gslr, best_gslr
# for svm
svm_score_sum = 0
svm_best_score = 0
global svm, best_svm
# for decision tree
dt_score_sum = 0
dt_best_score = 0
global dt, best_dt
# for naive_bayes
gnb_score_sum = 0
gnb_best_score = 0
```

```python
global gnb, best_gnb
# compare the performace of random forest classifer, logistic regression, svm,␣
↪decision tree, and naive_bayes without fine-tune
kf =KFold(n_splits=5)
for train_index, test_index in kf.split(data):
    train, test = data.iloc[train_index], data.iloc[test_index]
    x_train = train.drop(['label'], axis=1)
    y_train = train['label']
    x_test = test.drop(['label'], axis=1)
    y_test = test['label']
    # for random forest
    gsrf = RandomForestClassifier(random_state=1)
    gsrf.fit(x_train,y_train)
    y_predict = gsrf.predict(x_test)
    score = f1_score(y_test, y_predict, average='micro')
    if score > gsrf_best_score:
        gsrf_best_score = score
        best_gsrf = gsrf
    gsrf_score_sum = gsrf_score_sum + score
    # for logistic regression
    lr = LogisticRegression(penalty='l2', max_iter=10000)
    lr.fit(x_train, y_train)
    y_predict = lr.predict(x_test)
    score = f1_score(y_test, y_predict, average='micro')
    if score > lr_best_score:
        lr_best_score = score
        best_lr = lr
    lr_score_sum = lr_score_sum + score
    # for svm
    svm = SVC()
    svm.fit(x_train, y_train)
    y_predict = svm.predict(x_test)
    score = f1_score(y_test, y_predict, average='micro')
    if score > svm_best_score:
        svm_best_score = score
        best_svm = svm
    svm_score_sum = svm_score_sum + score
    # for decision tree
    dt = DecisionTreeClassifier(random_state=1)
    dt.fit(x_train, y_train)
    y_predict = dt.predict(x_test)
    score = f1_score(y_test, y_predict, average='micro')
    if score > dt_best_score:
        best_dt_score = score
        best_dt = dt
    dt_score_sum = dt_score_sum + score
    # for naive_bayes
    gnb = GaussianNB()
    gnb.fit(x_train, y_train)
    y_predict = gnb.predict(x_test)
    score = f1_score(y_test, y_predict, average='micro')
    if score > gnb_best_score:
        best_gnb_score = score
        best_gnb = gnb
    gnb_score_sum = gnb_score_sum + score
gsrf_score_sum/=5
lr_score_sum/=5
svm_score_sum/=5
dt_score_sum/=5
gnb_score_sum/=5
print('logistic regression f1 score: ', lr_score_sum)
print('svm f1 score:', svm_score_sum)
print('naive bayes classifer f1 score:', gnb_score_sum)
print('decision tree classifer f1 score:', dt_score_sum)
print('random forest classifier f1 score: ', gsrf_score_sum)
```

```
logistic regression f1 score:  0.667
svm f1 score: 0.524
naive bayes classifer f1 score: 0.633
decision tree classifer f1 score: 0.601
random forest classifier f1 score:  0.674
```

[14]:
```
# random forest classifer shows the best performance
# do fine tuning with grid search
gsrf_score_sum = 0
gsrf_best_score = 0
kf =KFold(n_splits=5)
for train_index, test_index in kf.split(data):
    train, test = data.iloc[train_index], data.iloc[test_index]
    x_train = train.drop(['label'], axis=1)
    y_train = train['label']
    x_test = test.drop(['label'], axis=1)
    y_test = test['label']
    # for random forest
    rf = RandomForestClassifier(random_state=1)
    prf = [{'n_estimators':[10,50,100,150],'max_depth':[3,6,9,12],'criterion':
 ↪['gini','entropy']}]
    gsrf = GridSearchCV(estimator = rf, param_grid = prf,scoring =␣
 ↪'accuracy',cv = 2)
    gsrf.fit(x_train,y_train)
    print(gsrf.best_params_)
    y_predict = gsrf.predict(x_test)
    score = f1_score(y_test, y_predict, average='micro')
    print(score)
    if score > gsrf_best_score:
        gsrf_best_score = score
        best_gsrf = gsrf
    gsrf_score_sum = gsrf_score_sum + score
gsrf_score_sum/=5
print('random forest classfier f1 score after fine tuning: ', gsrf_score_sum)
```

```
{'criterion': 'gini', 'max_depth': 3, 'n_estimators': 150}
0.705
{'criterion': 'entropy', 'max_depth': 3, 'n_estimators': 100}
0.675
{'criterion': 'entropy', 'max_depth': 3, 'n_estimators': 50}
0.715
{'criterion': 'gini', 'max_depth': 3, 'n_estimators': 10}
0.665
{'criterion': 'gini', 'max_depth': 9, 'n_estimators': 50}
0.645
random forest classfier f1 score after fine tuning:  0.6809999999999999
```

```
[ ]:
```