Data-X Spring 2019: Homework 06

Name : Zhang Jiaheng

SID : 3034453700

Course (IEOR 135/290) : IEOR 135

Machine Learning

In this homework, you will do some exercises with prediction. We will cover these algorithms in class, but this is for you to have some hands on with these in scikit-learn. You can refer - https://github.com/ikhlaqsidhu/data-x/blob/master/05a-tools-predicition-titanic/titanic.ipynb (https://github.com/ikhlaqsidhu/data-x/blob/master/05a-tools-predicition-titanic/titanic.ipynb)

Display all your outputs.

In [1]:

```
import numpy as np
import pandas as pd
```

In [2]:

```
 # machine learning libraries
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import Perceptron
from sklearn.tree import DecisionTreeClassifier
```

1. Read `diabetesdata.csv` **file into a pandas dataframe. About the data:**

1. **TimesPregnant**: Number of times pregnant
2. **glucoseLevel**: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. **BP**: Diastolic blood pressure (mm Hg)
4. **insulin**: 2-Hour serum insulin (mu U/ml)
5. **BMI**: Body mass index (weight in kg/(height in m)^2)
6. **pedigree**: Diabetes pedigree function
7. **Age**: Age (years)
8. **IsDiabetic**: 0 if not diabetic or 1 if diabetic)

In [3]:

```
#Read data & print the head
df = pd.read_csv('diabetesdata.csv')
df.head()
```

Out[3]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
0	6	148.0	72	0	33.6	0.627	50.0	1
1	1	NaN	66	0	26.6	0.351	31.0	0
2	8	183.0	64	0	23.3	0.672	NaN	1
3	1	NaN	66	94	28.1	0.167	21.0	0
4	0	137.0	40	168	43.1	2.288	33.0	1

2. Calculate the percentage of Null values in each column and display it.

In [5]:

```
for column in df:
    ratio = df[column].isna().sum()/len(df[column])
    print("{}: {}%".format(column, ratio*100))
```

```
TimesPregnant: 0.0%
glucoseLevel: 4.427083333333334%
BP: 0.0%
insulin: 0.0%
BMI: 0.0%
Pedigree: 0.0%
Age: 4.296875%
IsDiabetic: 0.0%
```

3. Split `data` into `train_df` and `test_df` with 15% as test.

In [6]:

```
from sklearn.model_selection import train_test_split
train_df, test_df = train_test_split(df, test_size=0.15)
print(train_df)
print(test_df)
```

```
     TimesPregnant  glucoseLevel  BP  insulin   BMI  Pedigree   Age  \
69               4         146.0  85      100  28.9     0.189  27.0
364              4         147.0  74      293  34.9     0.385  30.0
558             11         103.0  68        0  46.2     0.126  42.0
741              3         102.0  44       94  30.8     0.400  26.0
170              6         102.0  82        0  30.8     0.180  36.0
699              4         118.0  70        0  44.5     0.904  26.0
123              5         132.0  80        0  26.8     0.186  69.0
451              2         134.0  70        0  28.9     0.542  23.0
159             17         163.0  72      114  40.9     0.817  47.0
75               1           0.0  48        0  24.7     0.140   NaN
223              7         142.0  60      190  28.8     0.687  61.0
336              0         117.0   0        0  33.8     0.932  44.0
609              1         111.0  62      182  24.0     0.138  23.0
247              0         165.0  90      680  52.3     0.427  23.0
532              1          86.0  66       65  41.3     0.917  29.0
155              7         152.0  88        0  50.0     0.337  36.0
636              5         104.0  74        0  28.8     0.153  48.0
283              7         161.0  86        0  30.4     0.165  47.0
410              6         102.0  90        0  35.7     0.674  28.0
338              9         152.0  78      171  34.2     0.893  33.0
241              4          91.0  70       88  33.1     0.446  22.0
591              2         112.0  78      140  39.4     0.175  24.0
279              2         108.0  62      278  25.3     0.881  22.0
474              4         114.0  64        0  28.9     0.126  24.0
316              3          99.0  80       64  19.3     0.284  30.0
665              1         112.0  80      132  34.8     0.217  24.0
404              5         168.0  64        0  32.9     0.135  41.0
588              3         176.0  86      156  33.3     1.154  52.0
164              0         131.0  88        0  31.6     0.743  32.0
266              0         138.0   0        0  36.3     0.933  25.0
..             ...           ...  ..      ...   ...       ...   ...
425              4         184.0  78      277  37.0     0.264  31.0
491              2          89.0  90        0  33.5     0.292  42.0
12              10         139.0  80        0  27.1     1.441  57.0
51               1         101.0  50       36  24.2     0.526  26.0
169              3         111.0  90       78  28.4     0.495  29.0
753              0         181.0  88      510  43.3     0.222  26.0
59               0         105.0  64      142  41.5     0.173  22.0
713              0           NaN  58      291  26.4     0.352  21.0
321              3         112.0  74        0  31.6     0.197  25.0
417              4         144.0  82        0  38.5     0.554  37.0
729              2          92.0  52        0  30.1     0.141  22.0
312              2         155.0  74       96  26.6     0.433  27.0
526              1          97.0  64       82  18.2     0.299  21.0
380              1         107.0  72       82  30.8     0.821  24.0
286              5         155.0  84      545  38.7     0.619  34.0
252              2          90.0  80       55  24.4     0.249  24.0
743              9         140.0  94        0  32.7     0.734  45.0
539              3         129.0  92      155  36.4     0.968  32.0
429              1          95.0  82      180  35.0     0.233  43.0
457              5          86.0  68       71  30.2     0.364  24.0
293              1         128.0  48      194  40.5     0.613  24.0
196              1         105.0  58        0  24.3     0.187  21.0
528              0         117.0  66      188  30.8     0.493  22.0
37               9         102.0  76        0  32.9     0.665  46.0
301              2         144.0  58      135  31.6     0.422  25.0
485              0         135.0  68      250  42.3     0.365  24.0
471              0         137.0  70        0  33.2     0.170  22.0
165              6         104.0  74      156  29.9     0.722  41.0
```

```
47               2          71.0  70         0  28.0     0.586  22.0
617              2          68.0  62        15  20.1     0.257  23.0

     IsDiabetic
69            0
364           0
558           0
741           0
170           1
699           0
123           0
451           1
159           1
75            0
223           0
336           0
609           0
247           0
532           0
155           1
636           0
283           1
410           0
338           1
241           0
591           0
279           0
474           0
316           0
665           0
404           1
588           1
164           1
266           1
..          ...
425           1
491           0
12            0
51            0
169           0
753           1
59            0
713           0
321           1
417           1
729           0
312           1
526           0
380           0
286           0
252           0
743           1
539           1
429           1
457           0
293           1
196           0
528           0
37            1
301           1
485           1
```

```
471          0
165          1
47           0
617          0

[652 rows x 8 columns]
     TimesPregnant  glucoseLevel   BP  insulin   BMI  Pedigree   Age  \
377              1          87.0   60       75  37.2     0.509  22.0
465              0         124.0   56      105  21.8     0.452  21.0
619              0         119.0    0        0  32.4     0.141  24.0
662              8         167.0  106      231  37.6     0.165  43.0
675              6         195.0   70        0  30.9     0.328  31.0
512              9          91.0   68        0  24.2     0.200  58.0
589              0          73.0    0        0  21.1     0.342  25.0
261              3         141.0    0        0  30.0     0.761  27.0
398              3          82.0   70        0  21.1     0.389  25.0
346              1         139.0   46       83  28.7     0.654  22.0
433              2         139.0   75        0  25.6     0.167  29.0
46               1         146.0   56        0  29.7     0.564  29.0
420              1         119.0   88      170  45.3     0.507  26.0
622              6         183.0   94        0  40.8     1.461  45.0
407              0         101.0   62        0  21.9     0.336  25.0
573              2          98.0   60      120  34.7     0.198  22.0
245              9         184.0   85        0  30.0     1.213  49.0
258              1         193.0   50      375  25.9     0.655  24.0
464             10         115.0   98        0  24.0     1.022  34.0
635             13         104.0   72        0  31.2     0.465  38.0
146              9          57.0   80        0  32.8     0.096  41.0
453              2         119.0    0        0  19.6     0.832  72.0
544              1          88.0   78       76  32.0     0.365  29.0
34              10         122.0   78        0  27.6     0.512   NaN
565              2          95.0   54       88  26.1     0.748  22.0
616              6         117.0   96        0  28.7     0.157   NaN
615              3         106.0   72        0  25.8     0.207  27.0
652              5         123.0   74       77  34.1     0.269   NaN
402              5         136.0   84       88  35.0     0.286  35.0
472              0         119.0   66        0  38.8     0.259  22.0
..             ...           ...  ...      ...   ...       ...   ...
167              4         120.0   68        0  29.6     0.709  34.0
519              6         129.0   90      326  19.6     0.582  60.0
590             11         111.0   84        0  46.8     0.925  45.0
127              1         118.0   58       94  33.3     0.261  23.0
631              0         102.0   78       90  34.5     0.238  24.0
495              6         166.0   74        0  26.6     0.304  66.0
6                3          78.0   50       88  31.0     0.248  26.0
725              4           NaN   78        0  39.4     0.236  38.0
288              4          96.0   56       49  20.8     0.340   NaN
458             10         148.0   84      237  37.6     1.001  51.0
682              0          95.0   64      105  44.6     0.366  22.0
756              7         137.0   90        0  32.0     0.391  39.0
64               7         114.0   66        0  32.8     0.258  42.0
611              3         174.0   58      194  32.9     0.593  36.0
343              5         122.0   86        0  34.7     0.290  33.0
535              4         132.0    0        0  32.9     0.302  23.0
376              0          98.0   82       84  25.2     0.299  22.0
607              1          92.0   62       41  19.5     0.482  25.0
610              3         106.0   54      158  30.9     0.292  24.0
595              0         188.0   82      185  32.0     0.682  22.0
603              7         150.0   78      126  35.2     0.692  54.0
751              1         121.0   78       74  39.0     0.261  28.0
```

```
600            1         108.0   88         0  27.1     0.400  24.0
639            1         100.0   74        46  19.5     0.149  28.0
390            1         100.0   66       196  32.0     0.444  42.0
549            4         189.0  110         0  28.5     0.680  37.0
62             5          44.0   62         0  25.0     0.587  36.0
677            0          93.0   60         0  35.3     0.263  25.0
210            2          81.0   60         0  27.7     0.290  25.0
177            0         129.0  110       130  67.1     0.319  26.0

     IsDiabetic
377           0
465           0
619           1
662           1
675           1
512           0
589           0
261           1
398           0
346           0
433           0
46            0
420           0
622           0
407           0
573           0
245           1
258           0
464           0
635           1
146           0
453           0
544           0
34            0
565           0
616           0
615           0
652           0
402           1
472           0
..          ...
167           0
519           0
590           1
127           0
631           0
495           0
6             1
725           0
288           0
458           1
682           0
756           0
64            1
611           1
343           0
535           1
376           0
607           0
610           0
595           1
```

```
603         1
751         0
600         0
639         0
390         0
549         0
62          0
677         0
210         0
177         1

[116 rows x 8 columns]
```

4. Display the means of the features in train and test sets. Replace the null values in `train_df` and `test_df` with the mean of EACH feature column separately for train and test. Display head of the dataframes.

In [7]:

```
for column in train_df:
    print("%s, %f"%(column, train_df[column].mean()))
print('\ntrain_df complete')
print('starting mean for test_df\n')
for column2 in test_df:
    print("%s, %f"%(column2, test_df[column2].mean()))

###We can see which columns have null values in the above qn(2)
train_df.fillna(value={'glucoseLevel':120.849359}, inplace=True)
train_df.fillna(value={'Age':33.410543}, inplace=True)

test_df.fillna(value={'glucoseLevel':121.963636}, inplace=True)
test_df.fillna(value={'Age':33.027523}, inplace=True)

###Double Check
print('\nDouble Check\n')
for column in train_df:
    ratio = train_df[column].isna().sum()/len(train_df[column])
    print("{}: {}%".format(column, ratio*100))

print('\ntrain_df complete')
print('starting mean for test_df\n')

for column in test_df:
    ratio = test_df[column].isna().sum()/len(test_df[column])
    print("{}: {}%".format(column, ratio*100))
```

```
TimesPregnant, 3.838957
glucoseLevel, 121.329053
BP, 69.012270
insulin, 80.354294
BMI, 32.098620
Pedigree, 0.475106
Age, 33.435897
IsDiabetic, 0.365031

train_df complete
starting mean for test_df

TimesPregnant, 3.879310
glucoseLevel, 119.261261
BP, 69.629310
insulin, 76.681034
BMI, 31.396552
Pedigree, 0.453724
Age, 32.891892
IsDiabetic, 0.258621

Double Check

TimesPregnant: 0.0%
glucoseLevel: 0.0%
BP: 0.0%
insulin: 0.0%
BMI: 0.0%
Pedigree: 0.0%
Age: 0.0%
IsDiabetic: 0.0%

train_df complete
starting mean for test_df

TimesPregnant: 0.0%
glucoseLevel: 0.0%
BP: 0.0%
insulin: 0.0%
BMI: 0.0%
Pedigree: 0.0%
Age: 0.0%
IsDiabetic: 0.0%
```

```
/Users/jiahengzhang/anaconda3/lib/python3.7/site-packages/pandas/cor
e/generic.py:5434: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/panda
s-docs/stable/indexing.html#indexing-view-versus-copy
  self._update_inplace(new_data)
```

5. Split `train_df` & `test_df` into `X_train` , `Y_train` and `X_test` , `Y_test` . `Y_train` and `Y_test` should only have the column we are trying to predict, `IsDiabetic` .

In [8]:

```
X_train = train_df.iloc[:,:7]
Y_train = train_df['IsDiabetic']
X_test = test_df.iloc[:,:7]
Y_test = test_df['IsDiabetic']

print('\nX_train\n')
print(X_train)
print('\nY_train\n')
print(Y_train)
print('\nX_test\n')
print(X_test)
print('\nY_test\n')
print(Y_test)

print(X_train.shape, Y_train.shape, X_test.shape, Y_test.shape)
```

X_train

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age
69	4	146.000000	85	100	28.9	0.189	27.000000
364	4	147.000000	74	293	34.9	0.385	30.000000
558	11	103.000000	68	0	46.2	0.126	42.000000
741	3	102.000000	44	94	30.8	0.400	26.000000
170	6	102.000000	82	0	30.8	0.180	36.000000
699	4	118.000000	70	0	44.5	0.904	26.000000
123	5	132.000000	80	0	26.8	0.186	69.000000
451	2	134.000000	70	0	28.9	0.542	23.000000
159	17	163.000000	72	114	40.9	0.817	47.000000
75	1	0.000000	48	0	24.7	0.140	33.410543
223	7	142.000000	60	190	28.8	0.687	61.000000
336	0	117.000000	0	0	33.8	0.932	44.000000
609	1	111.000000	62	182	24.0	0.138	23.000000
247	0	165.000000	90	680	52.3	0.427	23.000000
532	1	86.000000	66	65	41.3	0.917	29.000000
155	7	152.000000	88	0	50.0	0.337	36.000000
636	5	104.000000	74	0	28.8	0.153	48.000000
283	7	161.000000	86	0	30.4	0.165	47.000000
410	6	102.000000	90	0	35.7	0.674	28.000000
338	9	152.000000	78	171	34.2	0.893	33.000000
241	4	91.000000	70	88	33.1	0.446	22.000000
591	2	112.000000	78	140	39.4	0.175	24.000000
279	2	108.000000	62	278	25.3	0.881	22.000000
474	4	114.000000	64	0	28.9	0.126	24.000000
316	3	99.000000	80	64	19.3	0.284	30.000000
665	1	112.000000	80	132	34.8	0.217	24.000000
404	5	168.000000	64	0	32.9	0.135	41.000000
588	3	176.000000	86	156	33.3	1.154	52.000000
164	0	131.000000	88	0	31.6	0.743	32.00

```
0000
266          0   138.000000   0      0  36.3     0.933  25.00
0000
..         ...          ...  ..    ...   ...       ...
...
425          4   184.000000  78    277  37.0     0.264  31.00
0000
491          2    89.000000  90      0  33.5     0.292  42.00
0000
12          10   139.000000  80      0  27.1     1.441  57.00
0000
51           1   101.000000  50     36  24.2     0.526  26.00
0000
169          3   111.000000  90     78  28.4     0.495  29.00
0000
753          0   181.000000  88    510  43.3     0.222  26.00
0000
59           0   105.000000  64    142  41.5     0.173  22.00
0000
713          0   120.849359  58    291  26.4     0.352  21.00
0000
321          3   112.000000  74      0  31.6     0.197  25.00
0000
417          4   144.000000  82      0  38.5     0.554  37.00
0000
729          2    92.000000  52      0  30.1     0.141  22.00
0000
312          2   155.000000  74     96  26.6     0.433  27.00
0000
526          1    97.000000  64     82  18.2     0.299  21.00
0000
380          1   107.000000  72     82  30.8     0.821  24.00
0000
286          5   155.000000  84    545  38.7     0.619  34.00
0000
252          2    90.000000  80     55  24.4     0.249  24.00
0000
743          9   140.000000  94      0  32.7     0.734  45.00
0000
539          3   129.000000  92    155  36.4     0.968  32.00
0000
429          1    95.000000  82    180  35.0     0.233  43.00
0000
457          5    86.000000  68     71  30.2     0.364  24.00
0000
293          1   128.000000  48    194  40.5     0.613  24.00
0000
196          1   105.000000  58      0  24.3     0.187  21.00
0000
528          0   117.000000  66    188  30.8     0.493  22.00
0000
37           9   102.000000  76      0  32.9     0.665  46.00
0000
301          2   144.000000  58    135  31.6     0.422  25.00
0000
485          0   135.000000  68    250  42.3     0.365  24.00
0000
471          0   137.000000  70      0  33.2     0.170  22.00
0000
165          6   104.000000  74    156  29.9     0.722  41.00
0000
```

```
47             2     71.000000  70        0  28.0     0.586  22.00
0000
617            2     68.000000  62       15  20.1     0.257  23.00
0000

[652 rows x 7 columns]

Y_train

69     0
364    0
558    0
741    0
170    1
699    0
123    0
451    1
159    1
75     0
223    0
336    0
609    0
247    0
532    0
155    1
636    0
283    1
410    0
338    1
241    0
591    0
279    0
474    0
316    0
665    0
404    1
588    1
164    1
266    1
      ..
425    1
491    0
12     0
51     0
169    0
753    1
59     0
713    0
321    1
417    1
729    0
312    1
526    0
380    0
286    0
252    0
743    1
539    1
429    1
457    0
293    1
```

```
196    0
528    0
37     1
301    1
485    1
471    0
165    1
47     0
617    0
Name: IsDiabetic, Length: 652, dtype: int64

X_test

     TimesPregnant  glucoseLevel   BP  insulin   BMI  Pedigree
Age
377              1     87.000000   60       75  37.2     0.509  22.0
00000
465              0    124.000000   56      105  21.8     0.452  21.0
00000
619              0    119.000000    0        0  32.4     0.141  24.0
00000
662              8    167.000000  106      231  37.6     0.165  43.0
00000
675              6    195.000000   70        0  30.9     0.328  31.0
00000
512              9     91.000000   68        0  24.2     0.200  58.0
00000
589              0     73.000000    0        0  21.1     0.342  25.0
00000
261              3    141.000000    0        0  30.0     0.761  27.0
00000
398              3     82.000000   70        0  21.1     0.389  25.0
00000
346              1    139.000000   46       83  28.7     0.654  22.0
00000
433              2    139.000000   75        0  25.6     0.167  29.0
00000
46               1    146.000000   56        0  29.7     0.564  29.0
00000
420              1    119.000000   88      170  45.3     0.507  26.0
00000
622              6    183.000000   94        0  40.8     1.461  45.0
00000
407              0    101.000000   62        0  21.9     0.336  25.0
00000
573              2     98.000000   60      120  34.7     0.198  22.0
00000
245              9    184.000000   85        0  30.0     1.213  49.0
00000
258              1    193.000000   50      375  25.9     0.655  24.0
00000
464             10    115.000000   98        0  24.0     1.022  34.0
00000
635             13    104.000000   72        0  31.2     0.465  38.0
00000
146              9     57.000000   80        0  32.8     0.096  41.0
00000
453              2    119.000000    0        0  19.6     0.832  72.0
00000
544              1     88.000000   78       76  32.0     0.365  29.0
00000
```

```
34            10   122.000000  78     0  27.6     0.512  33.0
27523
565            2    95.000000  54    88  26.1     0.748  22.0
00000
616            6   117.000000  96     0  28.7     0.157  33.0
27523
615            3   106.000000  72     0  25.8     0.207  27.0
00000
652            5   123.000000  74    77  34.1     0.269  33.0
27523
402            5   136.000000  84    88  35.0     0.286  35.0
00000
472            0   119.000000  66     0  38.8     0.259  22.0
00000
..           ...          ...  ...   ...   ...      ...
...
167            4   120.000000  68     0  29.6     0.709  34.0
00000
519            6   129.000000  90   326  19.6     0.582  60.0
00000
590           11   111.000000  84     0  46.8     0.925  45.0
00000
127            1   118.000000  58    94  33.3     0.261  23.0
00000
631            0   102.000000  78    90  34.5     0.238  24.0
00000
495            6   166.000000  74     0  26.6     0.304  66.0
00000
6              3    78.000000  50    88  31.0     0.248  26.0
00000
725            4   121.963636  78     0  39.4     0.236  38.0
00000
288            4    96.000000  56    49  20.8     0.340  33.0
27523
458           10   148.000000  84   237  37.6     1.001  51.0
00000
682            0    95.000000  64   105  44.6     0.366  22.0
00000
756            7   137.000000  90     0  32.0     0.391  39.0
00000
64             7   114.000000  66     0  32.8     0.258  42.0
00000
611            3   174.000000  58   194  32.9     0.593  36.0
00000
343            5   122.000000  86     0  34.7     0.290  33.0
00000
535            4   132.000000   0     0  32.9     0.302  23.0
00000
376            0    98.000000  82    84  25.2     0.299  22.0
00000
607            1    92.000000  62    41  19.5     0.482  25.0
00000
610            3   106.000000  54   158  30.9     0.292  24.0
00000
595            0   188.000000  82   185  32.0     0.682  22.0
00000
603            7   150.000000  78   126  35.2     0.692  54.0
00000
751            1   121.000000  78    74  39.0     0.261  28.0
00000
600            1   108.000000  88     0  27.1     0.400  24.0
```

```
00000
639             1    100.000000   74       46  19.5     0.149  28.0
00000
390             1    100.000000   66      196  32.0     0.444  42.0
00000
549             4    189.000000  110        0  28.5     0.680  37.0
00000
62              5     44.000000   62        0  25.0     0.587  36.0
00000
677             0     93.000000   60        0  35.3     0.263  25.0
00000
210             2     81.000000   60        0  27.7     0.290  25.0
00000
177             0    129.000000  110      130  67.1     0.319  26.0
00000

[116 rows x 7 columns]

Y_test

377    0
465    0
619    1
662    1
675    1
512    0
589    0
261    1
398    0
346    0
433    0
46     0
420    0
622    0
407    0
573    0
245    1
258    0
464    0
635    1
146    0
453    0
544    0
34     0
565    0
616    0
615    0
652    0
402    1
472    0
      ..
167    0
519    0
590    1
127    0
631    0
495    0
6      1
725    0
288    0
458    1
```

```
682    0
756    0
64     1
611    1
343    0
535    1
376    0
607    0
610    0
595    1
603    1
751    0
600    0
639    0
390    0
549    0
62     0
677    0
210    0
177    1
Name: IsDiabetic, Length: 116, dtype: int64
(652, 7) (652,) (116, 7) (116,)
```

6. Use this dataset to train perceptron, logistic regression and random forest models using 15% test split. Report training and test accuracies. Try different hyperparameter values for these models and see if you can improve your accuracies.

In [9]:

```
# 6a. Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
Y_pred = logreg.predict(X_test)
acc_logreg = sum(Y_pred == Y_test)/len(Y_test)*100

print('Logistic Regression labeling accuracy:', str(round(acc_logreg,2)),'%')
```

```
Logistic Regression labeling accuracy: 81.9 %

/Users/jiahengzhang/anaconda3/lib/python3.7/site-packages/sklearn/li
near_model/logistic.py:433: FutureWarning: Default solver will be ch
anged to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

In [13]:

```
# 6b. Perceptron
perceptron = Perceptron(penalty='elasticnet',alpha=0.01,max_iter=100,early_stopp
ing=True,validation_fraction=0.01)
perceptron.fit(X_train, Y_train)
acc_perceptron = perceptron.score(X_test, Y_test)

print('Perceptron labeling accuracy:', str(round(acc_perceptron*100,2)),'%')
```

```
Perceptron labeling accuracy: 68.1 %

/Users/jiahengzhang/anaconda3/lib/python3.7/site-packages/sklearn/li
near_model/stochastic_gradient.py:183: FutureWarning: max_iter and t
ol parameters have been added in Perceptron in 0.19. If max_iter is
set but tol is left unset, the default value for tol in 0.19 and 0.2
0 will be None (which is equivalent to -infinity, so it has no effec
t) but will change in 0.21 to 1e-3. Specify tol to silence this warn
ing.
  FutureWarning)
```

In [16]:

```
# 6c. Random Forest
random_forest = RandomForestClassifier(n_estimators=700)
random_forest.fit(X_train, Y_train)
acc_rf = random_forest.score(X_test, Y_test)

print('Random Forest labelling accuracy:', str(round(acc_rf*100,2)),'%')
```

```
Random Forest labelling accuracy: 81.9 %
```

7. For your logistic regression model -

a . Compute the log probability of classes in `IsDiabetic` for the first 10 samples of your train set and display it. Also display the predicted class for those samples from your logistic regression model trained before.

In [26]:

```
ten_samples_trainX = X_train[:10]
result_trainY = logreg.predict(ten_samples_trainX)
ten_samples_trainY = Y_train[:10]
log_likelihood_elements = logreg.predict_proba(ten_samples_trainX)
print('\nLog Probabilities\n')
print(log_likelihood_elements)
print('\nPredicted Values\n')
print(result_trainY)
```

```
Log Probabilities

[[0.64142553 0.35857447]
 [0.52323052 0.47676948]
 [0.4628212  0.5371788 ]
 [0.75387189 0.24612811]
 [0.79412337 0.20587663]
 [0.44732244 0.55267756]
 [0.61172379 0.38827621]
 [0.64484288 0.35515712]
 [0.06503828 0.93496172]
 [0.99072795 0.00927205]]

Predicted Values

[0 0 1 0 0 1 0 0 1 0]
```

b . Now compute the log probability of classes in `IsDiabetic` for the first 10 samples of your test set and display it. Also display the predicted class for those samples from your logistic regression model trained before. (using the model trained on the training set)

In [27]:

```
ten_samples_testX = X_test[:10]
result_testY = logreg.predict(ten_samples_testX)
ten_samples_testY = Y_test[:10]
log_likelihood_elements_test = logreg.predict_proba(ten_samples_testX)
print('\nLog Probabilities\n')
print(log_likelihood_elements_test)
print('\nPredicted Values\n')
print(result_testY)
```

```
Log Probabilities

[[0.83327598 0.16672402]
 [0.804337   0.195663  ]
 [0.59284604 0.40715396]
 [0.36107874 0.63892126]
 [0.1715084  0.8284916 ]
 [0.78030903 0.21969097]
 [0.88926905 0.11073095]
 [0.27708538 0.72291462]
 [0.92204697 0.07795303]
 [0.55430689 0.44569311]]

Predicted Values

[0 0 0 1 1 0 0 1 0 0]
```

c . What can you interpret from the log probabilities and the predicted classes?

As long as the log probability of a certain label is above 50%, the model would choose to output the label as the predicted label

There is still a significant amount of uncertainty in the prediction, as we can see that some of the probabilities for each label is similar (50+% to 40+%

8. Is mean imputation is the best type of imputation (as we did in 4.) to use? Why or why not? What are some other ways to impute the data?

Your answer here

In [29]:

```
Mean imputation has an advantage of keeping the same mean and
same sample size, but this could be inaccurate as this could introduce
inaccurate data if the value is actually significantly larger/lower
than the mean.
Some other methods include substitution: Impute the value from a
new individual who was not selected to be in the sample.
Hot Deck Implementation: A randomly chosen value from an individual
in the sample who has similar values on other variables.
Cold deck imputation: A systematically chosen value from an
individual who has similar values on other variables.
Regression imputation: The predicted value obtained by
regressing the missing variable on other variables.
Stochastic regression imputation: The predicted value
from a regression plus a random residual value.
Interpolation and extrapolation: An estimated value from other
observations from the same individual. It usually only works in
longitudinal data.
```

```
  File "<ipython-input-29-bdfba6ebf238>", line 1
    Mean imputation has an advantage of keeping the same mean and
                  ^
SyntaxError: invalid syntax
```

Extra Credit (2 pts) - MANDATORY for students enrolled in IEOR 290

9. Implement the K-Nearest Neighbours (https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm (https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm)) algorithm for k=1 from scratch in python (do not use KNN from existing libraries). KNN uses Euclidean distance to find nearest neighbors. Split your dataset into test and train as before. Also fill in the null values with mean of features as done earlier. Use this algorithm to predict values for 'IsDiabetic' for your test set. Display your accuracy.

In [ ]:

```

```