Data-X Spring 2019: Homework 06

Name :

SID :

Course (IEOR 135/290) :

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 [140]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [15]:

```
 # 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
from sklearn.model_selection import train_test_split
```

__ 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 [44]:

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

Out[44]:

	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 [45]:

```
for i in data:
    temp = data[i].isna().sum() / len(data[i])
    print("There are " , temp*100, "% NaN values in" , i, "Coloumn")
```

```
There are  0.0 % NaN values in TimesPregnant Coloumn
There are  4.427083333333334 % NaN values in glucoseLevel Coloumn
There are  0.0 % NaN values in BP Coloumn
There are  0.0 % NaN values in insulin Coloumn
There are  0.0 % NaN values in BMI Coloumn
There are  0.0 % NaN values in Pedigree Coloumn
There are  4.296875 % NaN values in Age Coloumn
There are  0.0 % NaN values in IsDiabetic Coloumn
```

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

In [46]:

```
train_df , test_df  = train_test_split( data, test_size=0.15, random_state=100)
```

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 [87]:

```
print("For train data")
for i in train_df:
    print("The mean of", i ,"Coloumn is ", train_df[i].mean())
```

```
For train data
The mean of TimesPregnant Coloumn is  3.7975460122699385
The mean of glucoseLevel Coloumn is  120.69131832797422
The mean of BP Coloumn is  68.8282208588957
The mean of insulin Coloumn is  78.74386503067484
The mean of BMI Coloumn is  31.81088957055214
The mean of Pedigree Coloumn is  0.46909662576687106
The mean of Age Coloumn is  33.37560192616372
The mean of IsDiabetic Coloumn is  0.348159509202454
```

In [89]:

```
print("For test data")
for i in test_df:
    print("The mean of", i ,"Coloumn is ", test_df[i].mean())
```

```
For test data
The mean of TimesPregnant Coloumn is  4.112068965517241
The mean of glucoseLevel Coloumn is  122.82142857142857
The mean of BP Coloumn is  70.66379310344827
The mean of insulin Coloumn is  85.73275862068965
The mean of BMI Coloumn is  33.013793103448286
The mean of Pedigree Coloumn is  0.4875000000000001
The mean of Age Coloumn is  33.232142857142854
The mean of IsDiabetic Coloumn is  0.35344827586206895
```

In [73]:

```
train_df['Age'].fillna(train_df['Age'].mean(), inplace = True)
train_df['glucoseLevel'].fillna(train_df['glucoseLevel'].mean(), inplace = True)
test_df['glucoseLevel'].fillna(test_df['glucoseLevel'].mean(), inplace = True)
test_df['Age'].fillna(test_df['Age'].mean(), inplace = True)
train_df.head()
```

Out[73]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
458	10	148.0	84	237	37.6	1.001	51.0	1
635	13	104.0	72	0	31.2	0.465	38.0	1
457	5	86.0	68	71	30.2	0.364	24.0	0
674	8	91.0	82	0	35.6	0.587	68.0	0
277	0	104.0	64	116	27.8	0.454	23.0	0

In [74]:

```
test_df.head()
```

Out[74]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
173	1	79.000000	60	48	43.5	0.678	23.0	0
253	0	86.000000	68	0	35.8	0.238	25.0	0
207	5	162.000000	104	0	37.7	0.151	52.0	1
737	8	122.821429	72	0	32.0	0.600	42.0	0
191	9	123.000000	70	94	33.1	0.374	40.0	0

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 [75]:

```
x_train = train_df.iloc[:,:6]
x_test = test_df.iloc[:,:6]
y_train = train_df["IsDiabetic"]
y_test = test_df["IsDiabetic"]
```

In [77]:

```
x_test.head()
```

Out[77]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree
173	1	79.000000	60	48	43.5	0.678
253	0	86.000000	68	0	35.8	0.238
207	5	162.000000	104	0	37.7	0.151
737	8	122.821429	72	0	32.0	0.600
191	9	123.000000	70	94	33.1	0.374

In [78]:

```
x_train.head()
```

Out[78]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree
458	10	148.0	84	237	37.6	1.001
635	13	104.0	72	0	31.2	0.465
457	5	86.0	68	71	30.2	0.364
674	8	91.0	82	0	35.6	0.587
277	0	104.0	64	116	27.8	0.454

In [79]:

```
y_train.head()
```

Out[79]:

```
458    1
635    1
457    0
674    0
277    0
Name: IsDiabetic, dtype: int64
```

In [80]:

```
y_test.head()
```

Out[80]:

```
173    0
253    0
207    1
737    0
191    0
Name: IsDiabetic, dtype: int64
```

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 [141]:

```
# 6a. Logistic Regression
model = LogisticRegression()
model.fit(x_train,y_train)
model.score(x_test,y_test)
```

Out[141]:

```
0.7586206896551724
```

In [158]:

```
# 6a. Logistic Regression
model = LogisticRegression(C = 0.1 ,intercept_scaling= 2, penalty='l2' ,max_iter=20,
model.fit(x_train,y_train)
model.score(x_test,y_test)
```

Out[158]:

```
0.75
```

In [142]:

```
# 6a. Logistic Regression
model = LogisticRegression(C = 1.1 ,intercept_scaling= 2, penalty='l2' ,max_iter=20)
model.fit(x_train,y_train)
model.score(x_test,y_test)
```

Out[142]:

```
0.7672413793103449
```

In [159]:

```
# 6b. Perceptron
per_model = Perceptron()
per_model.fit(x_train,y_train)
per_model.score(x_test,y_test)
```

Out[159]:

```
0.41379310344827586
```

In [162]:

```
# 6b. Perceptron
per_model = Perceptron(alpha=0.0001,eta0=1.0,max_iter=20)
per_model.fit(x_train,y_train)
per_model.score(x_test,y_test)
```

Out[162]:

```
0.5775862068965517
```

In [169]:

```
# 6b. Perceptron
per_model = Perceptron(alpha=9,eta0=9,max_iter=5000)
per_model.fit(x_train,y_train)
per_model.score(x_test,y_test)
```

Out[169]:

```
0.7068965517241379
```

In [170]:

```
# 6c. Random Forest
rand_model =RandomForestClassifier()
rand_model.fit(x_train,y_train)
rand_model.score(x_test,y_test)
```

Out[170]:

```
0.6724137931034483
```

In [172]:

```
# 6c. Random Forest
rand_model =RandomForestClassifier(min_samples_leaf=2,min_weight_fraction_leaf=0.1)
rand_model.fit(x_train,y_train)
rand_model.score(x_test,y_test)
```

Out[172]:

```
0.7155172413793104
```

In [180]:

```
# 6c. Random Forest
rand_model =RandomForestClassifier(min_samples_leaf=5,min_weight_fraction_leaf=0.1)
rand_model.fit(x_train,y_train)
rand_model.score(x_test,y_test)
```

Out[180]:

```
0.7586206896551724
```

*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 [196]:

```
model.predict_log_proba(x_train.iloc[:10])
```

Out[196]:

```
array([[-1.20379462, -0.35675132],
       [-0.64075999, -0.74843124],
       [-0.21541296, -1.64097214],
       [-0.33331252, -1.26070623],
       [-0.19369513, -1.73675468],
       [-0.94896232, -0.4896229 ],
       [-0.31450673, -1.30988512],
       [-0.47337014, -0.97524346],
       [-0.41498908, -1.07983221],
       [-0.135866  , -2.06325012]])
```

In [198]:

```
model.predict(x_train.iloc[:10])
```

Out[198]:

```
array([1, 0, 0, 0, 0, 1, 0, 0, 0, 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 [197]:

```
model.predict_log_proba(x_test.iloc[:10])
```

Out[197]:

```
array([[-0.25573511, -1.48875713],
       [-0.15199855, -1.95892112],
       [-0.65553079, -0.73223406],
       [-0.62855373, -0.76220283],
       [-0.68015026, -0.70631525],
       [-0.94059707, -0.49494353],
       [-2.35417296, -0.09978942],
       [-0.24323045, -1.53289732],
       [-1.33689668, -0.30470551],
       [-0.36855625, -1.17678671]])
```

In [199]:

```
model.predict(x_test.iloc[:10])
```

Out[199]:

```
array([0, 0, 0, 0, 0, 1, 1, 0, 1, 0])
```

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

It is noticed that whichever classes log probability value is higher(0 or 1) the model predicts that class.

This directly implies that the model predicts the class based on which class has a higher probability and does not significantly concider other factors.

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?

Mean Imputation might not be the best type of data impitation because it dosent realte to that row. There are better types like regression imputation (Predict the missing paramater using other paramaters) or substitution (finding a subject with similar paramaters and subtituting his value in the missing paramater). There are many other meathods like Hot deck imputation, cold deck imputation, Interpolation and extrapolation etc.

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 [ ]:

```

```