Data-X Spring 2019: Homework 06

Name :Jack Xie

SID :3032163590

Course (IEOR 135/290) :IEOR135

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 [2]:
```
import numpy as np
import pandas as pd
```

In [96]:
```
 # 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
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

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 [157]:
```
df = pd.read_csv('diabetesdata.csv')
df.head()
```

Out[157]:

	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 [158]:
```
df.isnull().sum()/768
```

Out[158]:
```
TimesPregnant    0.000000
glucoseLevel     0.044271
BP               0.000000
insulin          0.000000
BMI              0.000000
Pedigree         0.000000
Age              0.042969
IsDiabetic       0.000000
dtype: float64
```

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

In [180]:
```
from sklearn.model_selection import train_test_split
train_df, test_df = train_test_split(df, test_size=0.15)
```

In [182]:
```
train_df.head()
```

Out[182]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
72	13	126.0	90	0	43.4	0.583	42.0	1
232	1	79.0	80	37	25.4	0.583	22.0	0
67	2	109.0	92	0	42.7	0.845	54.0	0
438	1	97.0	70	0	18.2	0.147	21.0	0
49	7	105.0	0	0	0.0	0.305	24.0	0

In [203]:
```
test_df.head()
```

Out[203]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
302	5	77.0	82	42	35.8	0.156	35.0	0
51	1	101.0	50	36	24.2	0.526	26.0	0
189	5	139.0	80	160	31.6	0.361	25.0	1
672	10	68.0	106	49	35.5	0.285	47.0	0
333	12	106.0	80	0	23.6	0.137	44.0	0

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 [185]:
```
train_df.mean()
```

Out[185]:
```
TimesPregnant      3.794479
glucoseLevel     120.585209
BP                68.794479
insulin           79.783742
BMI               32.002914
Pedigree           0.467848
Age               33.134400
IsDiabetic         0.343558
dtype: float64
```

In [186]:
```
test_df.mean()
```

Out[186]:
```
TimesPregnant      4.129310
glucoseLevel     123.410714
BP                70.853448
insulin           79.887931
BMI               31.934483
Pedigree           0.494517
Age               34.600000
IsDiabetic         0.379310
dtype: float64
```

In [231]:
```
test_df.fillna(test_df.mean(), inplace = True)
test_df.head()
```

Out[231]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
302	5	77.0	82	42	35.8	0.156	35.0	0
51	1	101.0	50	36	24.2	0.526	26.0	0
189	5	139.0	80	160	31.6	0.361	25.0	1
672	10	68.0	106	49	35.5	0.285	47.0	0
333	12	106.0	80	0	23.6	0.137	44.0	0

In [232]:
```
train_df.fillna(train_df.mean(), inplace = True)
train_df.head()
```

Out[232]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
72	13	126.0	90	0	43.4	0.583	42.0	1
232	1	79.0	80	37	25.4	0.583	22.0	0
67	2	109.0	92	0	42.7	0.845	54.0	0
438	1	97.0	70	0	18.2	0.147	21.0	0
49	7	105.0	0	0	0.0	0.305	24.0	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 [284]:
```
X_train=train_df.iloc[:,:-1].values
Y_train=train_df['IsDiabetic'].values
X_test=test_df.iloc[:,:-1].values
Y_test=test_df['IsDiabetic'].values
```

In [285]:
```
X_train[:10]
```

Out[285]:
```
array([[1.30e+01, 1.26e+02, 9.00e+01, 0.00e+00, 4.34e+01, 5.83e-01,
        4.20e+01],
       [1.00e+00, 7.90e+01, 8.00e+01, 3.70e+01, 2.54e+01, 5.83e-01,
        2.20e+01],
       [2.00e+00, 1.09e+02, 9.20e+01, 0.00e+00, 4.27e+01, 8.45e-01,
        5.40e+01],
       [1.00e+00, 9.70e+01, 7.00e+01, 0.00e+00, 1.82e+01, 1.47e-01,
        2.10e+01],
       [7.00e+00, 1.05e+02, 0.00e+00, 0.00e+00, 0.00e+00, 3.05e-01,
        2.40e+01],
       [1.00e+00, 1.07e+02, 5.00e+01, 0.00e+00, 2.83e+01, 1.81e-01,
        2.90e+01],
       [4.00e+00, 1.29e+02, 6.00e+01, 2.31e+02, 2.75e+01, 5.27e-01,
        3.10e+01],
       [1.00e+00, 1.12e+02, 8.00e+01, 1.32e+02, 3.48e+01, 2.17e-01,
        2.40e+01],
       [4.00e+00, 1.10e+02, 6.60e+01, 0.00e+00, 3.19e+01, 4.71e-01,
        2.90e+01],
       [5.00e+00, 1.28e+02, 8.00e+01, 0.00e+00, 3.46e+01, 1.44e-01,
        4.50e+01]])
```

In [286]:
```
Y_train[:10]
```

Out[286]:
```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [287]:
```
X_test[:10]
```

Out[287]:
```
array([[5.00e+00, 7.70e+01, 8.20e+01, 4.20e+01, 3.58e+01, 1.56e-01,
        3.50e+01],
       [1.00e+00, 1.01e+02, 5.00e+01, 3.60e+01, 2.42e+01, 5.26e-01,
        2.60e+01],
       [5.00e+00, 1.39e+02, 8.00e+01, 1.60e+02, 3.16e+01, 3.61e-01,
        2.50e+01],
       [1.00e+01, 6.80e+01, 1.06e+02, 4.90e+01, 3.55e+01, 2.85e-01,
        4.70e+01],
       [1.20e+01, 1.06e+02, 8.00e+01, 0.00e+00, 2.36e+01, 1.37e-01,
        4.40e+01],
       [6.00e+00, 1.66e+02, 7.40e+01, 0.00e+00, 2.66e+01, 3.04e-01,
        6.60e+01],
       [0.00e+00, 1.04e+02, 6.40e+01, 6.40e+01, 3.36e+01, 5.10e-01,
        2.20e+01],
       [2.00e+00, 1.27e+02, 5.80e+01, 2.75e+02, 2.77e+01, 1.60e+00,
        2.50e+01],
       [2.00e+00, 1.55e+02, 5.20e+01, 5.40e+02, 3.87e+01, 2.40e-01,
        3.46e+01],
       [5.00e+00, 1.10e+02, 6.80e+01, 0.00e+00, 2.60e+01, 2.92e-01,
        3.00e+01]])
```

In [288]:
```
Y_test[:10]
```

Out[288]:
```
array([0, 0, 1, 0, 0, 0, 1, 0, 1, 0])
```

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 [289]:
```
# 6a. Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
Y_pred_test = logreg.predict(X_test)
Y_pred_train = logreg.predict(X_train)
acc_test = sum(Y_pred_test == Y_test)/len(Y_test)*100
acc_train = sum(Y_pred_train == Y_train)/len(Y_train)*100

print('Logistic Regression test accuracy:', str(round(acc_test,2)),'%')
print('Logistic Regression train accuracy:', str(round(acc_train,2)),'%'
)
```

```
Logistic Regression test accuracy: 77.59 %
Logistic Regression train accuracy: 77.91 %
```

In [290]:
```
# 6b. Perceptron
perceptron = Perceptron()
perceptron.fit(X_train, Y_train)
acc_perceptron_test = perceptron.score(X_test, Y_test)
acc_perceptron_train = perceptron.score(X_train, Y_train)

print('Perceptron test accuracy:', str(round(acc_perceptron_test*100,2
)),'%')
print('Perceptron train accuracy:', str(round(acc_perceptron_train*100,2
)),'%')
```

```
Perceptron test accuracy: 42.24 %
Perceptron train accuracy: 36.96 %
```

In [291]:
```
# 6c. Random Forest
random_forest = RandomForestClassifier(n_estimators=800)
random_forest.fit(X_train, Y_train)
acc_rf_test = random_forest.score(X_test, Y_test)
acc_rf_train = random_forest.score(X_train, Y_train)

print('K-Nearest Neighbors test accuracy:', str(round(acc_rf_test*100,2
)),'%')
print('K-Nearest Neighbors train accuracy:', str(round(acc_rf_train*100,
2)),'%')
```

```
K-Nearest Neighbors test accuracy: 76.72 %
K-Nearest Neighbors train accuracy: 100.0 %
```

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 [292]:
```
logreg.predict_log_proba(X_train[:10])
```

Out[292]:
```
array([[-1.2774632 , -0.32676048],
       [-0.06908121, -2.70681427],
       [-0.49793781, -0.93593934],
       [-0.06254286, -2.80301167],
       [-0.17738185, -1.8168308 ],
       [-0.18916969, -1.75820505],
       [-0.412471  , -1.08474605],
       [-0.17807741, -1.81325466],
       [-0.32498975, -1.2820596 ],
       [-0.46391567, -0.99105894]])
```

In [293]:
```
logreg.predict(X_train)[:10]
```

Out[293]:
```
array([1, 0, 0, 0, 0, 0, 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 [294]:
```
logreg.predict_log_proba(X_test[:10])
```

Out[294]:
```
array([[-0.13225625, -2.08841335],
       [-0.16713344, -1.87136584],
       [-0.46258962, -0.9933094 ],
       [-0.16016921, -1.91054038],
       [-0.34212509, -1.23876908],
       [-1.05152496, -0.4298672 ],
       [-0.19583454, -1.72680495],
       [-0.63695128, -0.75269008],
       [-0.73319175, -0.65464462],
       [-0.23888724, -1.54883058]])
```

In [295]:
```
logreg.predict(X_test)[:10]
```

Out[295]:
```
array([0, 0, 0, 0, 0, 1, 0, 0, 1, 0])
```

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

if the probability closer to zero the higher chance for the model to be correctly perdicted.

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?

No, the logistic regrassion model will not accurately predict the labels, if the null is filled by mean. Becuase the classification will not produce any more diversity of the outputs, which will cause the inaccuracy of the model.

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 [296]:
```
def knn(X_train, X_test, Y_test, Y_train):
    distances = np.zeros([len(X_test), len(X_train)])
    for i,x in enumerate(X_test):
        for j,y in enumerate(X_train):
            distances[i,j] = np.sqrt(((y-x)**2).sum())
    neighbors = np.nanargmin(distances, axis = 1)
    print("\nPredicted Class Values:\n",Y_train[neighbors])
    print('\nThe accuracy of the Solver is:', str(round((label == Y_test
).sum()/len(label)*100,2)),'%')
```

In [ ]:
```
knn(X_train, X_test, Y_train, Y_test)
```