Data-X Spring 2019: Homework 06

Name : Ziyu Li

SID : 3034331915

Course (IEOR 135/290) : IEOR 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 [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')
print(df.head())
```

```
   TimesPregnant  glucoseLevel  BP  insulin   BMI  Pedigree   Age  IsDiab
etic
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 [4]:
```
df.isnull().mean()*100
```

Out[4]:
```
TimesPregnant    0.000000
glucoseLevel     4.427083
BP               0.000000
insulin          0.000000
BMI              0.000000
Pedigree         0.000000
Age              4.296875
IsDiabetic       0.000000
dtype: float64
```

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

In [5]:
```
from sklearn.model_selection import train_test_split
train_df, test_df = train_test_split(df, test_size=0.15)
```

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 [6]:
```
# display the means of the features in train and test sets
train_df.mean()
test_df.mean()
```

Out[6]:
```
TimesPregnant      3.991379
glucoseLevel     121.135135
BP                67.068966
insulin           72.715517
BMI               31.214655
Pedigree           0.485888
Age               31.396396
IsDiabetic         0.293103
dtype: float64
```

In [7]:
```
# replace the null value with the mean
train_df=train_df.fillna(train_df.mean())
test_df=test_df.fillna(test_df.mean())
print(train_df.head())
print(test_df.head())
```

```
     TimesPregnant  glucoseLevel  BP  insulin   BMI  Pedigree   Age  \
461              1          71.0  62        0  21.8     0.416  26.0
285              7         136.0  74      135  26.0     0.647  51.0
154              8         188.0  78        0  47.9     0.137  43.0
739              1         102.0  74        0  39.5     0.293  42.0
191              9         123.0  70       94  33.1     0.374  40.0

     IsDiabetic
461           0
285           0
154           1
739           1
191           0
     TimesPregnant  glucoseLevel  BP  insulin   BMI  Pedigree        Age
\
201              1         138.0  82        0  40.1     0.236  28.000000
95               6         144.0  72      228  33.9     0.255  40.000000
497              2          81.0  72       76  30.1     0.547  25.000000
751              1         121.0  78       74  39.0     0.261  28.000000
299              8         112.0  72        0  23.6     0.840  31.396396

     IsDiabetic
201           0
95            0
497           0
751           0
299           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 [8]:
```
X_train = train_df.drop('IsDiabetic', axis=1)
Y_train = train_df['IsDiabetic']
X_test = test_df.drop('IsDiabetic', axis=1)
Y_test = test_df['IsDiabetic']
```

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)
log_accuracy_train = logreg.score(X_train, Y_train)
log_accuracy_test = logreg.score(X_test, Y_test)

print('Logistic Regression training accuracy is : ', log_accuracy_train)
print('Logistic Regression ttest accuracy is : ', log_accuracy_test)
```

```
Logistic Regression training accuracy is :  0.7730061349693251
Logistic Regression ttest accuracy is :  0.8017241379310345
```

In [10]:
```
# hyperparameters tuning for logistic regression when c=100,penalty='l1'
logreg100 = LogisticRegression(penalty='l1',C=100)
logreg100.fit(X_train, Y_train)
logreg_train_acc = logreg100.score(X_train, Y_train)
logreg_test_acc = logreg100.score(X_test, Y_test)
print ('logistic regression training acuracy is ',logreg_train_acc)
print('logistic regression test accuracy is ',logreg_test_acc)
```

```
logistic regression training acuracy is  0.7760736196319018
logistic regression test accuracy is  0.8103448275862069
```

In [11]:
```
# hyperparameters tuning for logistic regression when c=0.1,penalty='l1'
logreg= LogisticRegression(penalty='l1',C=0.1)
logreg.fit(X_train, Y_train)
logreg_train_acc = logreg.score(X_train, Y_train)
logreg_test_acc = logreg.score(X_test, Y_test)
print ('logistic regression training acuracy is ',logreg_train_acc)
print('logistic regression test accuracy is ',logreg_test_acc)
```

```
logistic regression training acuracy is  0.7484662576687117
logistic regression test accuracy is  0.8275862068965517
```

In [12]:
```
# 6b. Perceptron
perceptron = Perceptron()
perceptron.fit(X_train, Y_train)
perceptron_train_acc = perceptron.score(X_train, Y_train)
perceptron_test_acc = perceptron.score(X_test, Y_test)
print ('Perceptron training acuracy is ',perceptron_train_acc)
print('Perceptron test accuracy is ',perceptron_test_acc)
```

```
Perceptron training acuracy is  0.6395705521472392
Perceptron test accuracy is  0.6810344827586207
```

```
/Users/Jade/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/st
ochastic_gradient.py:128: FutureWarning: max_iter and tol parameters have
been added in <class 'sklearn.linear_model.perceptron.Perceptron'> in 0.1
9. If both are left unset, they default to max_iter=5 and tol=None. If to
l is not None, max_iter defaults to max_iter=1000. From 0.21, default max
_iter will be 1000, and default tol will be 1e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
```

In [13]:
```
# with hyperparameter penalty='elasticnet', max_iter=50
perceptron = Perceptron(penalty='elasticnet',max_iter=50)
perceptron.fit(X_train, Y_train)
perceptron_train_acc = perceptron.score(X_train, Y_train)
perceptron_test_acc = perceptron.score(X_test, Y_test)
print ('Perceptron training acuracy is ',perceptron_train_acc)
print('Perceptron test accuracy is ',perceptron_test_acc)
```

```
Perceptron training acuracy is  0.3604294478527607
Perceptron test accuracy is  0.29310344827586204
```

In [14]:
```
# 6c. Random Forest
random_forest = RandomForestClassifier(n_estimators=500)
random_forest.fit(X_train, Y_train)
rf_train_acc = random_forest.score(X_train, Y_train)
rf_test_acc = random_forest.score(X_test, Y_test)
print ('Random Forest training acuracy is ',rf_train_acc)
print('Random Forest test accuracy is ',rf_test_acc)
```

```
Random Forest training acuracy is  1.0
Random Forest test accuracy is  0.8017241379310345
```

In [15]:
```
# with hyperparameters
random_forest = RandomForestClassifier(n_estimators=400,max_features=5,max_c
random_forest.fit(X_train, Y_train)
rf_train_acc = random_forest.score(X_train, Y_train)
rf_test_acc = random_forest.score(X_test, Y_test)
print ('Random Forest training acuracy is ',rf_train_acc)
print('Random Forest test accuracy is ',rf_test_acc)
```

```
Random Forest training acuracy is  0.9187116564417178
Random Forest test accuracy is  0.7931034482758621
```

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 [22]:
```
X_train_sample=X_train.iloc[:10,:]
y_prob=logreg.predict_log_proba(X_train_sample)
print(y_prob)
y_pred=logreg.predict(X_train_sample)
print(y_pred)
```

```
[[-0.09799533 -2.37143302]
 [-0.57651404 -0.82520208]
 [-1.91697005 -0.15905652]
 [-0.31140209 -1.31833413]
 [-0.72100137 -0.66604787]
 [-0.59949675 -0.79648285]
 [-0.3830692  -1.14496744]
 [-1.07366047 -0.41817844]
 [-0.19142523 -1.74744424]
 [-1.09132756 -0.40912747]]
[0 0 1 0 1 0 0 1 0 1]
```

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 [23]:
```
X_test_sample=X_test.iloc[:10,:]
y_prob=logreg.predict_log_proba(X_test_sample)
print(y_prob)
y_pred=logreg.predict(X_test_sample)
print(y_pred)
```

```
[[-0.49104521 -0.94671491]
 [-0.79943195 -0.59708143]
 [-0.1606618  -1.90770935]
 [-0.38803879 -1.1344033 ]
 [-0.36974773 -1.17411826]
 [-0.8114563  -0.587366  ]
 [-0.54126079 -0.87230732]
 [-0.31746395 -1.30192719]
 [-0.24756324 -1.51731849]
 [-0.66215505 -0.72513063]]
[0 1 0 0 0 1 0 0 0 0]
```

c . What can you interpret from the log probabilities and the predicted classes? Since the probability of x is from 0 to 1, the log probability should be from negative infinity to 0.

The threshold for this logistic regression model is 0.5, when the probability of a sample greater than 0.5, it will be classified as class 1 while if smaller than 0.5 it will be classified as class 0. Thus, when looking at the log probability, we just need to compare two columns for the sample, the greater one is the classification outcome for the sample.

Type **Markdown** and LaTeX: α^2

Type **Markdown** and LaTeX: α^2

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?

The mean imputation is not the best type of imputation because it will bias the standard error. There are some other ways to impute the data like EM imputation, Hot deck imputation, Cold deck imputation, Regression imputation.

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)](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 [21]:

```
#Define Euclidean distances
import math
import operator

def Euclideandist(x,xi, length):
    d = 0.0
    for i in range(length):
        d += pow(float(x[i])- float(xi[i]),2)
    return math.sqrt(d)

#Getting neighbours
def getNeighbors(trainingSet, testInstance, k):
    distances = []
    length = len(testInstance)-1
    for x in range(len(trainingSet)):
        dist = Euclideandist(testInstance, trainingSet[x], length)
        distances.append((trainingSet[x], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors

# get response
def getResponse(neighbors):
    classVotes = {}
    for x in range(len(neighbors)):
        response = neighbors[x][-1]
        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
    sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), rev
    return sortedVotes[0][0]

# get accuracy
def getAccuracy(testSet, predictions):
    correct = 0
    for x in range(len(testSet)):
        if testSet[x][-1] == predictions[x]:
            correct += 1
    return (correct/float(len(testSet))) * 100.0

# generate predictions for k=1 testSet = test_df.values.tolist()
testSet=test_df.values.tolist()
trainingSet = train_df.values.tolist()

predictions=[]
k=1
for x in range(len(testSet)):
    neighbors = getNeighbors(trainingSet, testSet[x], k)
    result = getResponse(neighbors)
    predictions.append(result)

accuracy = getAccuracy(testSet, predictions)
print('Accuracy: ' + repr(accuracy) + '%')
```

```
Accuracy: 70.6896551724138%
```

In [ ]: