hw6_sp2019

March 6, 2019

1 Data-X Spring 2019: Homework 06

1.1 Name : Isha Mangal

1.2 SID : 3031911156

1.3 Course (IEOR 135/290) :

1.3.1 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

Display all your outputs.

In [47]:
```
import numpy as np
import pandas as pd
```

In [48]:
```
 # 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 [49]:
```python
#Read data & print the head
df = pd.read_csv('diabetesdata.csv')
df.head()
```

Out[49]:

	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 [50]:
```python
df.isnull().sum() / len(df)
```

Out[50]:
```
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 [51]:
```python
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 [52]:
```python
# means of the features in train set
means_train = {}
for col in train_df:
    means_train = train_df.mean()
print (means_train)
```

```
TimesPregnant      3.849693
glucoseLevel     121.186795
BP                68.854294
insulin           80.343558
BMI               31.907209
Pedigree           0.467644
Age               33.278400
IsDiabetic         0.349693
dtype: float64
```

In [53]:
```
# means of the features in test set
means_test = {}
for col in test_df:
    means_test = test_df.mean()
print (means_test)
```

```
TimesPregnant      3.818966
glucoseLevel     120.079646
BP                70.517241
insulin           76.741379
BMI               32.472414
Pedigree           0.495664
Age               33.781818
IsDiabetic         0.344828
dtype: float64
```

In [54]:
```
# replace null values in train_df with the mean of each feature
train_df = train_df.fillna(value=means_train)
train_df.head()
```

Out[54]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age
326	1	122.000000	64	156	35.1	0.692	30.0000
712	10	129.000000	62	0	41.2	0.441	33.2784
708	9	121.186795	78	0	32.8	0.148	45.0000
429	1	95.000000	82	180	35.0	0.233	43.0000
721	1	114.000000	66	200	38.1	0.289	21.0000

	IsDiabetic
326	1
712	1
708	1
429	1
721	0

In [55]:
```
# replace null values in test_df with the mean of each feature
test_df = test_df.fillna(value=means_test)
test_df.head()
```

Out[55]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age
603	7	150.0	78	126	35.2	0.692	54.0
356	1	125.0	50	167	33.3	0.962	28.0
51	1	101.0	50	36	24.2	0.526	26.0
41	7	133.0	84	0	40.2	0.696	37.0
59	0	105.0	64	142	41.5	0.173	22.0

	IsDiabetic
603	1
356	1

```
51            0
41            0
59            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 [56]:
```python
# split train_df into x_train & y_train
x_train = train_df.drop('IsDiabetic', axis=1)
y_train = train_df['IsDiabetic']

# split test_df into x_test & y_test
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 [57]:
```python
# 6a. Logistic Regression

from sklearn import linear_model

logreg_model = linear_model.LogisticRegression(C=1e5)
logreg_model.fit(x_train, y_train)

train_accuracy = logreg_model.score(x_train, y_train)
print ('Logistic Regression Training Accuracy:', train_accuracy)

test_accuracy = logreg_model.score(x_test, y_test)
print ('Logistic Regression Test Accuracy:', test_accuracy)
```

```
Logistic Regression Training Accuracy: 0.777607361963
Logistic Regression Test Accuracy: 0.801724137931
```

In [58]:
```python
# 6b. Perceptron

percep_model = linear_model.Perceptron()
percep_model.fit(x_train, y_train)

train_accuracy = percep_model.score(x_train, y_train)
print ('Perceptron Training Accuracy:', train_accuracy)

test_accuracy = percep_model.score(x_test, y_test)
print ('Perceptron Test Accuracy:', test_accuracy)
```

```
Perceptron Training Accuracy: 0.665644171779
Perceptron Test Accuracy: 0.663793103448
```

```
/Users/ishamangal/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/stochastic_gradie
  "and default tol will be 1e-3." % type(self), FutureWarning)
```

In [59]:
```python
# 6c. Random Forest

from sklearn import ensemble

rfr_model = ensemble.RandomForestRegressor()
rfr_model.fit(x_train, y_train)

train_accuracy = rfr_model.score(x_train, y_train)
print ('Random Forest Regressor Training Accuracy:', train_accuracy)

test_accuracy = rfr_model.score(x_test, y_test)
print ('Random Forest Regressor Test Accuracy:', test_accuracy)
```

```
Random Forest Regressor Training Accuracy: 0.855398874545
Random Forest Regressor Test Accuracy: 0.214328947368
```

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 [60]:
```python
# log probability of classes of train set
# you DON'T do it on y_train because it specifically regards the features
logreg_model.predict_log_proba(x_train.head(10))
```

Out[60]:
```
array([[-0.38681611, -1.13698723],
       [-1.47377335, -0.26014412],
       [-0.575304  , -0.82675591],
       [-0.11344396, -2.23263215],
       [-0.24259448, -1.53521029],
       [-0.12160209, -2.16718613],
       [-0.10704935, -2.28751259],
       [-0.36684528, -1.1806367 ],
       [-0.08484233, -2.50908196],
       [-1.234013  , -0.34407178]])
```

In [61]:
```python
# display the predicted class for those samples
logreg_model.predict(x_train.head(10))
```

Out[61]:
```
array([0, 1, 0, 0, 0, 0, 0, 0, 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 [62]:
```
# log probability of classes of test set
# you DON'T do it on y_test because it specifically regards the features
logreg_model.predict_log_proba(x_test.head(10))
```

Out[62]:
```
array([[-1.41644338, -0.27783106],
       [-0.49390465, -0.94222147],
       [-0.09058012, -2.44646871],
       [-1.21923   , -0.35020674],
       [-0.22445045, -1.60422734],
       [-0.11552348, -2.21548724],
       [-0.22203223, -1.61389557],
       [-0.11589361, -2.21246991],
       [-1.01209925, -0.45170047],
       [-0.13487466, -2.07008883]])
```

In [63]:
```
# display the predicted class for those samples
logreg_model.predict(x_test.head(10))
```

Out[63]:
```
array([1, 0, 0, 1, 0, 0, 0, 0, 1, 0])
```

c . What can you interpret from the log probabilities and the predicted classes?
For the train set, it shows that we pick the class 0 over 1 more frequently.
For the test set, it shows that we pick both class 0 and 1 an equal amount of times.

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 is useful in this scenario because null (NaN) values are numeric that work best if they are averaged. In general though, some disadvantages of this type of imputation include reducing variance in the dataset and not preserving the relationships among variables. Other ways to impute the data involve median imputation, multiple imputation, imputation of categorical variables, or hot-deck imputation.

1.4 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) 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.