hw6_sp2019

March 3, 2019

1 Data-X Spring 2019: Homework 06

1.1 Name : Shrey Samdani

1.2 SID : 3032000414

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 [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 [4]:
```
df.isnull().sum()/df.shape[0]
```

Out[4]:
```
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 [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]:
```
print(train_df.mean())
print()
print(test_df.mean())
print()
train_df = train_df.fillna(train_df.mean())
test_df = test_df.fillna(test_df.mean())

print(train_df.head())
print()
print(test_df.head())
```

```
TimesPregnant      3.812883
glucoseLevel     121.903069
BP                69.231595
insulin           81.188650
```

```
BMI               31.904908
Pedigree           0.474874
Age               33.626603
IsDiabetic         0.351227
dtype: float64

TimesPregnant      4.025862
glucoseLevel     116.243478
BP                68.396552
insulin           71.991379
BMI               32.485345
Pedigree           0.455026
Age               31.819820
IsDiabetic         0.336207
dtype: float64

     TimesPregnant  glucoseLevel  BP  insulin   BMI  Pedigree        Age  \
518             13          76.0  60        0  32.8     0.180  41.000000
622              6         183.0  94        0  40.8     1.461  45.000000
130              4         173.0  70      168  29.7     0.361  33.000000
712             10         129.0  62        0  41.2     0.441  33.626603
275              2         100.0  70       57  40.5     0.677  25.000000

     IsDiabetic
518           0
622           0
130           1
712           1
275           0

     TimesPregnant  glucoseLevel  BP  insulin   BMI  Pedigree       Age  \
536              0         105.0  90        0  29.6     0.197  46.00000
692              2         121.0  70       95  39.1     0.886  23.00000
598              1         173.0  74        0  36.8     0.088  31.81982
197              3         107.0  62       48  22.9     0.678  23.00000
375             12         140.0  82      325  39.2     0.528  58.00000

     IsDiabetic
536           0
692           0
598           1
197           1
375           1
```

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 [7]: X_train = train_df.iloc[:,:-1]
```

```
Y_train = train_df.iloc[:,-1]

X_test = test_df.iloc[:,:-1]
Y_test = test_df.iloc[:,-1]
```

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 [8]: # 6a. Logistic Regression

        logreg = LogisticRegression()
        logreg.fit(X_train, Y_train)
        print("TRAINING ACCURACY:",logreg.score(X_train,Y_train))
        print("VALIDATION ACCURACY:",logreg.score(X_test,Y_test))
```

```
TRAINING ACCURACY: 0.7791411042944786
VALIDATION ACCURACY: 0.7327586206896551


/Users/shreysamdani/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433
  FutureWarning)
```

```
In [9]: # 6b. Perceptron
        perceptron = Perceptron()
        perceptron.fit(X_train, Y_train)
        print("TRAINING ACCURACY:",perceptron.score(X_train,Y_train))
        print("VALIDATION ACCURACY:",perceptron.score(X_test,Y_test))
```

```
TRAINING ACCURACY: 0.6426380368098159
VALIDATION ACCURACY: 0.6637931034482759


/Users/shreysamdani/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/stochastic_grad
  FutureWarning)
```

```
In [12]: # 6c. Random Forest
         random_forest = RandomForestClassifier(n_estimators=500)
         random_forest.fit(X_train, Y_train)
         print("TRAINING ACCURACY:",random_forest.score(X_train,Y_train))
         print("VALIDATION ACCURACY:",random_forest.score(X_test,Y_test))
```

```
TRAINING ACCURACY: 1.0
VALIDATION ACCURACY: 0.7413793103448276
```

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 [25]:
```
print(logreg.predict_log_proba(X_train.head(10)))
print(logreg.predict(X_train.head(10)))
```

```
[[-0.39073917 -1.12873113]
 [-2.13868876 -0.12534694]
 [-0.97480211 -0.47363743]
 [-1.24120266 -0.34113404]
 [-0.30224963 -1.34382328]
 [-1.6239362  -0.21955159]
 [-1.15887555 -0.37664281]
 [-0.6841134  -0.70226332]
 [-0.20462773 -1.68713269]
 [-0.18978842 -1.75523923]]
[0 1 1 1 0 1 1 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 [67]:
```
print(logreg.predict_log_proba(X_test.head(10)))
print(logreg.predict(X_test.head(10)))
```

```
[[-0.10290063 -2.32500066]
 [-0.50297466 -0.92818397]
 [-0.87795155 -0.53722683]
 [-0.20332609 -1.69288529]
 [-1.31452892 -0.3127959 ]
 [-0.2900843  -1.37912212]
 [-0.23562671 -1.56100754]
 [-1.36034768 -0.29648276]
 [-0.49973155 -0.93316608]
 [-0.2113928  -1.6578724 ]]
[0 0 1 0 1 0 0 1 0 0]
```

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

The log probability that is closer to 0 is the class that will be predicted. Also, since most of the log probabilities have a significant difference between the classes, the model should be fairly strong at predicting the correct class

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?

It is not necessarily the best type of imputation. Filling in the NA values with the mean might not be consistent with the other features. One alternative is to perform regression on the other features to predict a value for the missing feature. Another alternative is to pick a random data point in the same column that has similar values in other columns and use that value to fill the NA value.

In [ ]:

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.

In [68]:
```
class knn:

    def fit(self, X_train, Y_train):
        self.X = X_train
        self.Y = Y_train

    def predict(self, X_test):
        euclidean  = lambda x,y: sum((x-y)**2)**0.5
        n = X_train.shape[0]
        indices = list(range(n))
        ans = []
        for point in X_test.iterrows():
            minIndex = min(indices, key = lambda x: euclidean(point[1],self.X.iloc[x]
            ans.append(self.Y.iloc[minIndex])
        return ans

    def score(self, X_test, Y_test):
        return sum(self.predict(X_test) == Y_test)/len(Y_test)

KNN = knn()
KNN.fit(X_train, Y_train)
KNN.score(X_test, Y_test)
```

Out[68]: 0.6724137931034483

In [ ]: