hw6_MachineLearning_fall2018

October 7, 2018

1 Data-X Fall 2018: Homework 06

1.0.1 Machine Learning

Authors: Sana Iqbal (Part 1, 2, 3)
In this homework, you will do some exercises with prediction.

```
In [1]: import numpy as np
        import pandas as pd
```

```
In [2]:  # machine learning libraries
        from sklearn.linear_model import LogisticRegression
        from sklearn.svm import SVC, LinearSVC
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.ensemble import AdaBoostClassifier
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.naive_bayes import GaussianNB
        from sklearn.linear_model import Perceptron
        from sklearn.linear_model import SGDClassifier
        from sklearn.tree import DecisionTreeClassifier
        import xgboost as xgb
```

1.1 Part 1

__ 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 it
        data = pd.read_csv("diabetesdata.csv")
```

```
data.info()
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 8 columns):
TimesPregnant    768 non-null int64
glucoseLevel     734 non-null float64
BP               768 non-null int64
insulin          768 non-null int64
BMI              768 non-null float64
Pedigree         768 non-null float64
Age              735 non-null float64
IsDiabetic       768 non-null int64
dtypes: float64(4), int64(4)
memory usage: 48.1 KB
```

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 NaN values in each column.

In [4]:
```
NullsPerColumn = pd.DataFrame()
NullsPerColumn['Percentage Null'] = data.isna().sum() / data.shape[0]
NullsPerColumn
```

Out[4]:

	Percentage Null
TimesPregnant	0.000000
glucoseLevel	0.044271
BP	0.000000
insulin	0.000000
BMI	0.000000
Pedigree	0.000000
Age	0.042969
IsDiabetic	0.000000

In [5]:
```
###RUN THIS CELL BUT DO NOT ALTER IT
assert all(NullsPerColumn.columns == ['Percentage Null'])
assert NullsPerColumn['Percentage Null'][-2] ==  0.04296875
```

3. Calculate the TOTAL percent of ROWS with NaN values in the dataframe (make sure values are floats).

In [6]:
```
PercentNull =  data.isna().max(axis="columns").sum()/data.shape[0]
PercentNull
```

```
Out[6]: 0.08333333333333333
```

4. Split data into train_df and test_df with 15% test split.

```
In [7]: #split values
        from sklearn.model_selection import train_test_split
        train_df, test_df = train_test_split(data, test_size=0.15)

In [8]: ###RUN THIS CELL BUT DO NOT ALTER IT
        np.testing.assert_almost_equal(float(len(train_df))/float(len(data)), 0.84895833333333
        np.testing.assert_almost_equal(float(len(test_df))/float(len(data)), 0.151041666666666
```

5. Replace the Nan values in train_df and test_df with the mean of EACH feature.

```
In [9]: train_df = train_df.fillna(train_df.mean())
        test_df = test_df.fillna(test_df.mean())

In [10]: ###RUN THIS CELL BUT DO NOT ALTER IT
         assert sum(train_df.isnull().sum()) == 0
         assert sum(test_df.isnull().sum()) == 0
```

6. 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 [11]: X_train = train_df.drop(columns="IsDiabetic")
         Y_train = train_df["IsDiabetic"]
         X_test  = test_df.drop(columns="IsDiabetic")
         Y_test = test_df["IsDiabetic"]

In [12]: ###RUN THIS CELL BUT DO NOT ALTER IT
         assert [X_train.shape, Y_train.shape, X_test.shape,Y_test.shape] == [(652, 7), (652,)
```

7.Use this dataset to train perceptron, logistic regression and random forest models using 15% test split. Report training and test accuracies.

```
In [13]: # Logistic Regression

         logreg = LogisticRegression()
         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 ('logreg training acuracy= ',logreg_train_acc)
         print('logreg test accuracy= ',logreg_test_acc)

logreg training acuracy=  0.7745398773006135
logreg test accuracy=  0.7327586206896551


c:\users\resident\documents\data\lib\site-packages\sklearn\linear_model\logistic.py:432: Futur
  FutureWarning)
```

```
In [14]: # 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= ',perceptron_train_acc)
         print('perceptron test accuracy= ',perceptron_test_acc)
```

```
perceptron training acuracy=  0.5521472392638037
perceptron test accuracy=  0.5689655172413793
```

```
c:\users\resident\documents\data\lib\site-packages\sklearn\linear_model\stochastic_gradient.py
  FutureWarning)
```

```
In [15]: # Adaboost
         adaboost = AdaBoostClassifier()
         adaboost.fit(X_train, Y_train)
         adaboost_train_acc = adaboost.score(X_train, Y_train)
         adaboost_test_acc = adaboost.score(X_test, Y_test)
         print ('adaboost training acuracy= ',adaboost_train_acc)
         print('adaboost test accuracy= ',adaboost_test_acc)
```

```
adaboost training acuracy=  0.8220858895705522
adaboost test accuracy=  0.6810344827586207
```

```
In [16]: # Random Forest

         random_forest = RandomForestClassifier()
         random_forest.fit(X_train, Y_train)
         random_forest_train_acc = random_forest.score(X_train, Y_train)
         random_forest_test_acc = random_forest.score(X_test, Y_test)
         print('random_forest training acuracy= ',random_forest_train_acc)
         print('random_forest test accuracy= ',random_forest_test_acc)
```

```
random_forest training acuracy=  0.9938650306748467
random_forest test accuracy=  0.7586206896551724
```

```
c:\users\resident\documents\data\lib\site-packages\sklearn\ensemble\forest.py:248: FutureWarni
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

8. Is mean imputation is the best type of imputation to use? Why or why not? What are some other ways to impute the data?

Mean imputation is one of the easiest imputation to use. It is not the best type of imputation since it creates a bias for this value of the feature. There are many other ways to do imputation, among others: - mean imputation from a specific number of groups (create N groups so that each row belongs to one group, for each row, replace nan values by the mean of the feature in all the examples of the group) - create an estimator (like linear regression) to predict the missing features with the others.

1.2 Part 2

1.Add columns `BMI_band__` & **`Pedigree_band`** to **`Data`** by cutting **`BMI`** & **`Pedigree`** into 3 intervals. PRINT the first 5 rows of__`data`**.**

```
In [17]: data['BMI_band'] = pd.cut(data['BMI'], 3)
         data['Pedigree_band'] = pd.cut(data['Pedigree'], 3)
         data.head()
```

```
Out[17]:    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

                    BMI_band    Pedigree_band
         0  (22.367, 44.733]  (0.0757, 0.859]
         1  (22.367, 44.733]  (0.0757, 0.859]
         2  (22.367, 44.733]  (0.0757, 0.859]
         3  (22.367, 44.733]  (0.0757, 0.859]
         4  (22.367, 44.733]    (1.639, 2.42]
```

1a. Print the category intervals for `BMI_band__` & `__Pedigree_band`.

```
In [18]: print('BMI_Band_Interval: {}'.format(data['BMI_band'].unique()))
         print('Pedigree_Band_Interval: {}'.format(data['Pedigree_band'].unique()))
```

```
BMI_Band_Interval: [(22.367, 44.733], (-0.0671, 22.367], (44.733, 67.1]]
Categories (3, interval[float64]): [(-0.0671, 22.367] < (22.367, 44.733] < (44.733, 67.1]]
Pedigree_Band_Interval: [(0.0757, 0.859], (1.639, 2.42], (0.859, 1.639]]
Categories (3, interval[float64]): [(0.0757, 0.859] < (0.859, 1.639] < (1.639, 2.42]]
```

2. Group `data__` by **`Pedigree_band`** & determine ratio of diabetic in each band.__

```
In [19]: pedigree_DiabeticRatio = data.groupby("Pedigree_band", as_index=False).mean()
         pedigree_DiabeticRatio["IsDiabetic"]
```

```
Out[19]: 0    0.327007
         1    0.540541
         2    0.444444
         Name: IsDiabetic, dtype: float64
```

2a. Group `data__` by **`BMI_band`** & determine ratio of diabetic in each band.__

In [20]:
```python
BMI_DiabeticRatio = data.groupby("BMI_band", as_index=False).mean()
BMI_DiabeticRatio["IsDiabetic"]
```

Out[20]:
```
0    0.039216
1    0.358297
2    0.611111
Name: IsDiabetic, dtype: float64
```

In [21]:
```python
###RUN THIS CELL BUT DO NOT ALTER IT
assert BMI_DiabeticRatio['IsDiabetic'][1] == 0.35829662261380324
assert pedigree_DiabeticRatio['IsDiabetic'][1] == 0.5405405405405406
```

3. Convert these features - 'BP','insulin','BMI' and 'Pedigree' into categorical values by mapping different bands of values of these features to integers 0,1,2.

HINT: USE pd.cut with bin=3 to create 3 bins

In [22]:
```python
list_features = ['BP', 'insulin', 'BMI', 'Pedigree']
for feature in list_features:
    bins = pd.cut(data[feature], 3, labels=False)
    data[feature] = bins

data = data.drop(columns=["BMI_band", "Pedigree_band"])
```

In [23]:
```python
###RUN THIS CELL BUT DO NOT ALTER IT
assert sum(data['insulin'])==49
assert sum(data['BMI'])==753
assert sum(data['Pedigree'])==92
```

4. Now consider the original dataset again, instead of generalizing the NAN values with the mean of the feature we will try assigning values to NANs based on some hypothesis. For example for age we assume that the relation between BMI and BP of people is a reflection of the age group. We can have 9 types of BMI and BP relations and our aim is to find the median age of each of that group:

Your Age guess matrix will look like this:

BMI	0	1	2
BP			
0	a00	a01	a02
1	a10	a11	a12
2	a20	a21	a22

Create a guess_matrix for NaN values of *'Age'* (using 'BMI' and 'BP') and *'glucoseLevel'* (using 'BP' and 'Pedigree') for the given dataset and assign values accordingly to the NaNs in 'Age' or *'glucoseLevel'* .

Refer to how we guessed age in the titanic notebook in the class.

In [24]:
```python
age_matrix, glucose_matrix = np.zeros((3, 3)), np.zeros((3, 3))
age_df, glucose_df = pd.DataFrame(), pd.DataFrame()
```

```python
matrices = [age_matrix, glucose_matrix]
dataframes = [age_df, glucose_df]
columns = ["Age", "glucoseLevel"]
group_features = ["BMI", "Pedigree"]
for matrix, df, column, feature in zip(matrices, dataframes, columns, group_features)
    for i in range(0, 3):
        for j in range(0, 3):
            guess_df = data[(data['BP'] == i) \
                        &(data[feature] == j)][column].dropna()

            # Extract the median age for this group
            # (less sensitive) to outliers
            guess_value = guess_df.median()

            # Convert random age float to int
            matrix[i, j] = int(guess_value)

    df = pd.DataFrame(matrix)
    df.columns.name = feature
    df.index.name = 'BP'
    print("-"*35)
    print('Guess table for {}:\n'.format(column), df)

    print ('\nAssigning median {value} values to NAN {value} values in the dataset...
    print()
    for i in range(0, 3):
        for j in range(0, 3):
            data.loc[ (data[column].isnull()) & (data['BP'] == i) \
                    & (data[feature] == j), column] = matrix[i,j]


    data[column] = data[column].astype(int)
```

```
-----------------------------------
Guess table for Age:
 BMI     0     1     2
BP
0    24.0  29.0  33.0
1    25.0  29.0  32.0
2    55.0  37.0  31.0

Assigning median Age values to NAN Age values in the dataset...

-----------------------------------
Guess table for glucoseLevel:
 Pedigree      0      1      2
BP
```

```
0          115.0  127.0  137.0
1          112.0  115.0  149.0
2          133.0  129.0  159.0

Assigning median glucoseLevel values to NAN glucoseLevel values in the dataset...
```

5. Now, convert 'glucoseLevel' and 'Age' features also to categorical variables of 4 categories each. PRINT the head of data__ __

In [25]:
```
list_features = ['glucoseLevel', 'Age']
for feature in list_features:
    bins = pd.cut(data[feature], 4, labels=False)
    data[feature] = bins
data.head()
```

Out[25]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
0	6	2	1	0	1	0	1	1
1	1	2	1	0	1	0	0	0
2	8	3	1	0	1	0	0	1
3	1	2	1	0	1	0	0	0
4	0	2	0	0	1	2	0	1

6.Use this dataset (with all features in categorical form) to train perceptron, logistic regression and random forest models using 15% test split. Report training and test accuracies.

In [26]:
```
train_df, test_df = train_test_split(data, test_size=0.15)
X_train = train_df.drop(columns="IsDiabetic")
Y_train = train_df["IsDiabetic"]
X_test = test_df.drop(columns="IsDiabetic")
Y_test = test_df["IsDiabetic"]
X_train.shape, Y_train.shape, X_test.shape
```

Out[26]: ((652, 7), (652,), (116, 7))

In [27]:
```
# Logistic Regression
logreg = LogisticRegression()
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 ('logreg training acuracy= ',logreg_train_acc)
print('logreg test accuracy= ',logreg_test_acc)
```

```
logreg training acuracy=  0.7515337423312883
logreg test accuracy=  0.6896551724137931


c:\users\resident\documents\data\lib\site-packages\sklearn\linear_model\logistic.py:432: Futur
  FutureWarning)
```

```
In [28]: # 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= ',perceptron_train_acc)
         print('perceptron test accuracy= ',perceptron_test_acc)
```

```
perceptron training acuracy=  0.7024539877300614
perceptron test accuracy=  0.6896551724137931
```

```
c:\users\resident\documents\data\lib\site-packages\sklearn\linear_model\stochastic_gradient.py
  FutureWarning)
```

```
In [95]: # Random Forest
         random_forest = RandomForestClassifier()
         random_forest.fit(X_train, Y_train)
         random_forest_train_acc = random_forest.score(X_train, Y_train)
         random_forest_test_acc = random_forest.score(X_test, Y_test)
         print ('random_forest training acuracy= ',random_forest_train_acc)
         print('random_forest test accuracy= ',random_forest_test_acc)
```

```
random_forest training acuracy=  0.8696319018404908
random_forest test accuracy=  0.7155172413793104
```

```
c:\users\resident\documents\data\lib\site-packages\sklearn\ensemble\forest.py:248: FutureWarni
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```