hw6_MachineLearning_fall2018

October 4, 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
        import warnings
        warnings.filterwarnings('ignore')
```

```
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.head(5)
```

```
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 = (data.isnull().sum()/len(data)).to_frame('Percentage Null')
        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.isnull().any(axis=1).sum()/len(data)).astype(float)
        print('The total percent of rows with NaN values in the dataframe is',PercentNull)
```

```
The total percent of rows with NaN values in the dataframe is 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,random_state=100)
```

```
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)), 0.15104166666666666
```

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("IsDiabetic", axis=1)
         Y_train = train_df["IsDiabetic"]
         X_test  = test_df.drop("IsDiabetic", axis=1)
         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=  0.75
```

```
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=  0.4156441717791411
perceptron test accuracy=  0.45689655172413796
```

```
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=  0.8358895705521472
adaboost test accuracy=  0.7241379310344828
```

```
In [16]: # Random Forest
         random_forest = RandomForestClassifier(n_estimators=500)
         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=  1.0
random_forest test accuracy=  0.7241379310344828
```

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?

Not exactly. The mean value of a certain feature doesn't change, but the relationships among other variables do. And that's usually what people are interested in, which are biased or simply erased by mean imputation.

Other ways include - median imputation - mode imputation - multiple imputation ('eyeball' some obivious relationships among features and determine the value of null position according to values of other related variables) - EM imputation (An iterative procedure in which it uses other variables to impute a value (Expectation), then checks whether that is the value most likely (Maximization). If not, it re-imputes a more likely value)

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]: # YOUR CODE HERE
         data['BMI_band']=pd.cut(data['BMI'],3)
         data['Pedigree_band']=pd.cut(data['Pedigree'],3)
         data.head(5)
```

```
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: ' + str(pd.unique(data['BMI_band'].values)))
         print('Pedigree_Band_Interval: ' + str(pd.unique(data['Pedigree_band'].values)))
```

```
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]: # YOUR CODE HERE
         pedigree_DiabeticRatio = data.groupby('Pedigree_band',as_index=False).mean()
         pedigree_DiabeticRatio
```

```
Out[19]:      Pedigree_band  TimesPregnant  glucoseLevel         BP     insulin  \
         0  (0.0757, 0.859]       3.870073    120.191424  68.757664   75.702190
         1   (0.859, 1.639]       3.932432    125.500000  72.486486  105.878378
         2    (1.639, 2.42]       1.222222    145.000000  67.777778  177.222222

                  BMI  Pedigree        Age  IsDiabetic
         0  31.659562  0.384975  33.307339    0.327007
         1  34.739189  1.090770  34.375000    0.540541
         2  34.755556  1.997333  28.555556    0.444444
```

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

```
In [20]: # YOUR CODE HERE
         BMI_DiabeticRatio = data.groupby('BMI_band',as_index=False).mean()
         BMI_DiabeticRatio
```

```
Out[20]:             BMI_band  TimesPregnant  glucoseLevel         BP     insulin  \
         0  (-0.0671, 22.367]       2.568627    102.297872  54.803922   36.823529
         1   (22.367, 44.733]       3.964758    121.767228  69.566814   81.449339
         2     (44.733, 67.1]       3.388889    132.470588  80.638889  109.472222

                  BMI  Pedigree        Age  IsDiabetic
         0  16.194118  0.380255  30.591837    0.039216
         1  32.284875  0.475261  33.537634    0.358297
         2  48.844444  0.537639  33.800000    0.611111
```

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

```
In [22]: data.head(5)
```

```
Out[22]:    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]
```

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 [23]: # YOUR CODE HERE
         data['BP']=pd.cut(data['BP'],bins=3,labels=np.arange(3))
         data['insulin']=pd.cut(data['insulin'],bins=3,labels=np.arange(3))
         data['BMI']=pd.cut(data['BMI'],bins=3,labels=np.arange(3))
         data['Pedigree']=pd.cut(data['Pedigree'],bins=3,labels=np.arange(3))
```

```
In [24]: ###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 [25]: # YOUR CODE HERE
```

```
guess_ages = np.zeros((3,3),dtype=int)

for i in range(0, 3):
    for j in range(0,3):
        guess_df = data[(data['BP'] == i) \
                    &(data['BMI'] == j)]['Age'].dropna()

        age_guess = guess_df.median()

            # Convert random age float to int
        guess_ages[i,j] = int(age_guess)


print('Guess_Age table:\n',guess_ages)

for i in range(0, 3):
    for j in range(0, 3):
        data.loc[ (data.Age.isnull()) & (data.BP == i) \
                & (data.BMI == j),'Age'] = guess_ages[i,j]


data['Age'] = data['Age'].astype(int)

guess_glucoseLevel = np.zeros((3,3),dtype=int)

for i in range(0, 3):
    for j in range(0,3):
        guess_df_g = data[(data['Pedigree'] == i) \
                    &(data['BP'] == j)]['glucoseLevel'].dropna()

        glucoseLevel_guess = guess_df_g.median()

            # Convert random age float to int
        guess_glucoseLevel[i,j] = int(glucoseLevel_guess)


print('Guess_glucoseLevel table:\n',guess_glucoseLevel)

for i in range(0, 3):
    for j in range(0, 3):
        data.loc[ (data.glucoseLevel.isnull()) & (data.Pedigree == i) \
                & (data.BP == j),'glucoseLevel'] = guess_glucoseLevel[i,j]


data['glucoseLevel'] = data['glucoseLevel'].astype(int)

data.head()
```

```
Guess_Age table:
 [[24 29 33]
 [25 29 32]
 [55 37 31]]
Guess_glucoseLevel table:
 [[115 112 133]
 [127 115 129]
 [137 149 159]]
```

Out[25]:

```
   TimesPregnant  glucoseLevel BP insulin BMI Pedigree  Age  IsDiabetic  \
0              6           148  1       0   1        0   50           1
1              1           112  1       0   1        0   31           0
2              8           183  1       0   1        0   29           1
3              1           112  1       0   1        0   21           0
4              0           137  0       0   1        2   33           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]
```

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

In [26]:

```
# YOUR CODE HERE
data['Age']=pd.cut(data['Age'],bins=4,labels=np.arange(4))
data['glucoseLevel']=pd.cut(data['glucoseLevel'],bins=4,labels=np.arange(4))
data.head()
```

Out[26]:

```
   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

          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]
```

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 [27]: train_df, test_df = train_test_split(data, test_size=0.15,random_state=100)
         X_train = train_df.iloc[:,1:7]
         Y_train = train_df.iloc[:,7]
         X_test  = test_df.iloc[:,1:7]
         Y_test= test_df.iloc[:,7]
         X_train.shape, Y_train.shape, X_test.shape
```

```
Out[27]: ((652, 6), (652,), (116, 6))
```

```
In [28]: # 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=  0.7469325153374233
logreg test accuracy=  0.7155172413793104
```

```
In [29]: # 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=  0.602760736196319
perceptron test accuracy=  0.5603448275862069
```

```
In [30]: # Random Forest
         random_forest = RandomForestClassifier(n_estimators=500)
         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=  0.8174846625766872
random_forest test accuracy=  0.6810344827586207
```