Data-X Fall 2018: Homework 06

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
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/weight_boosting.p
y:29: DeprecationWarning: numpy.core.umath_tests is an internal NumPy mod
ule and should not be imported. It will be removed in a future NumPy rele
ase.
  from numpy.core.umath_tests import inner1d
```

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('/Users/chelseayang/Downloads/diabetesdata.csv')
print(data)
data.columns
```

```
    TimesPregnant  glucoseLevel  BP  insulin   BMI  Pedigree   Age  \
0               6         148.0  72        0  33.6     0.627  50.0
1               1           NaN  66        0  26.6     0.351  31.0
2               8         183.0  64        0  23.3     0.672   NaN
3               1           NaN  66       94  28.1     0.167  21.0
4               0         137.0  40      168  43.1     2.288  33.0
5               5         116.0  74        0  25.6     0.201  30.0
6               3          78.0  50       88  31.0     0.248  26.0
7              10         115.0   0        0  35.3     0.134  29.0
8               2         197.0  70      543  30.5     0.158  53.0
9               8           NaN  96        0   0.0     0.232  54.0
10              4         110.0  92        0  37.6     0.191   NaN
11             10         168.0  74        0  38.0     0.537  34.0
12             10         139.0  80        0  27.1     1.441  57.0
13              1           NaN  60      846  30.1     0.398  59.0
14              5         166.0  72      175  25.8     0.587  51.0
15              7         100.0   0        0  30.0     0.484  32.0
16              0           NaN  84      230  45.8     0.551  31.0
17              7         107.0  74        0  29.6     0.254  31.0
```

2. Calculate the percentage of NaN values in each column.

In [4]:
```
NullsPerColumn = pd.DataFrame(np.random.randint(low=0, high=1, size=(1, 8)),c


for i in range(0,7):
    perc=round(data.iloc[:,i].isnull().sum()/768,4)*100
    perc=str(perc)+str('%')
    NullsPerColumn.iloc[[0],i]=perc




print(NullsPerColumn)
```

```
  TimesPregnant glucoseLevel    BP insulin   BMI Pedigree   Age  IsDiabet
ic
0          0.0%        4.43%  0.0%    0.0%  0.0%     0.0%  4.3%
0
```

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]:
```
null_val=data.isnull().sum(axis=1)

count_row=0

for i in range(0,768):
    if null_val[i]>=1:
        count_row=count_row+1

percent_null=count_row/768*1.0

percentNull = round(percent_null,4)*100

PercentNull=str(percentNull)+str('%')

print(PercentNull)
```

```
8.33%
```

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=15)
```

In [8]:
```
###RUN THIS CELL BUT DO NOT ALTER IT
np.testing.assert_almost_equal(float(len(train_df))/float(len(data)), 0.8489
np.testing.assert_almost_equal(float(len(test_df))/float(len(data)), 0.15104
```

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

In [9]:
```
guess_val_train = []
guess_val_test=[]

for i in range(0,8):
    guess_train_i=train_df.iloc[:,i].mean()
    guess_val_train.append(guess_train_i)


# replace he Nan values in  train_df with the mean of EACH feature
for i in range(0,8):
    train_df.iloc[:,i]=train_df.iloc[:,i].fillna(guess_val_train[i])

# Replace the nan values in test_df
for i in range(0,8):
    guess_test_i=test_df.iloc[:,i].mean()
    guess_val_test.append(guess_test_i)


# replace he Nan values in  test_df with the mean of EACH feature
for i in range(0,8):
    test_df.iloc[:,i]=test_df.iloc[:,i].fillna(guess_val_test[i])

print(train_df)
print(test_df)
```

```
/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.py:543: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydat
a.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  self.obj[item] = s
```

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.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=  0.7822085889570553
logreg test accuracy=  0.7413793103448276
```

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=  0.651840490797546
perceptron test accuracy=  0.6982758620689655

/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/stochastic_gr
adient.py:128: FutureWarning: max_iter and tol parameters have been added
in <class 'sklearn.linear_model.perceptron.Perceptron'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not No
ne, 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 [15]:
```
# Adaboost
adaboost = AdaBoostClassifier(n_estimators=100)
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=  0.8404907975460123
adaboost test accuracy=  0.75
```

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=  0.75
```

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?

Your answer here

In [17]:
```
#No, because sometimes the mean of this column is not representative of the
#and if there are too many outliers, the mean will become very bad to
#impute nan value. All in all, firstly, we have to understand what the nan
#and why they are nan, and then decide to use what kind of method.
```

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 [18]:
```
# YOUR CODE HERE
#raise NotImplementedError()

data['BMI_band']=pd.cut(data['BMI'],3)

data['Pedigree_band']=pd.cut(data['Pedigree'],3)


data.head()
```

Out[18]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_band	Pedigree_
0	6	148.0	72	0	33.6	0.627	50.0	1	(22.367, 44.733]	(0.0757, 0
1	1	NaN	66	0	26.6	0.351	31.0	0	(22.367, 44.733]	(0.0757, 0
2	8	183.0	64	0	23.3	0.672	NaN	1	(22.367, 44.733]	(0.0757, 0
3	1	NaN	66	94	28.1	0.167	21.0	0	(22.367, 44.733]	(0.0757, 0
4	0	137.0	40	168	43.1	2.288	33.0	1	(22.367, 44.733]	(1.639,

1a. Print the category intervals for _`*BMI*band` & _`*Pedigree*band`.

In [19]:
```
print('BMI_Band_Interval: ' + str(data['BMI_band'].unique()))

print('Pedigree_Band_Interval: ' + str(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 [20]:
```
# YOUR CODE HERE
#raise NotImplementedError()

diabetic_sum = data.groupby('Pedigree_band')['IsDiabetic'].sum()

count=data.groupby('Pedigree_band')['IsDiabetic'].count()

pedigree_DiabeticRatio=diabetic_sum/count*1.0

print(pedigree_DiabeticRatio)
```

```
Pedigree_band
(0.0757, 0.859]    0.327007
(0.859, 1.639]     0.540541
(1.639, 2.42]      0.444444
Name: IsDiabetic, dtype: float64
```

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

In [21]:
```
# YOUR CODE HERE
#raise NotImplementedError()


diabetic_sum2 = data.groupby('BMI_band')['IsDiabetic'].sum()

count2=data.groupby('BMI_band')['IsDiabetic'].count()

BMI_DiabeticRatio=diabetic_sum2/count2*1.0

print(BMI_DiabeticRatio)
```

```
BMI_band
(-0.0671, 22.367]    0.039216
(22.367, 44.733]     0.358297
(44.733, 67.1]       0.611111
Name: IsDiabetic, dtype: float64
```

In [22]:
```
###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 [23]:
```
# YOUR CODE HERE
#raise NotImplementedError()

data['BP']=pd.cut(data['BP'],3,labels=[0,1,2])

data['insulin']=pd.cut(data['insulin'],3,labels=[0,1,2])

data['BMI']=pd.cut(data['BMI'],3,labels=[0,1,2])

data['Pedigree']=pd.cut(data['Pedigree'],3,labels=[0,1,2])

data.head()
```

Out[23]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_band	Pedigree_
0	6	148.0	1	0	1	0	50.0	1	(22.367, 44.733]	(0.0757, 0
1	1	NaN	1	0	1	0	31.0	0	(22.367, 44.733]	(0.0757, 0
2	8	183.0	1	0	1	0	NaN	1	(22.367, 44.733]	(0.0757, 0
3	1	NaN	1	0	1	0	21.0	0	(22.367, 44.733]	(0.0757, 0
4	0	137.0	0	0	1	2	33.0	1	(22.367, 44.733]	(1.639,

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
#raise NotImplementedError()

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

for i in range(0,3):
    for j in range(0,3):
        guess_df_age=data[(data['BMI']==i) & (data['BP']==j)]['Age'].dropna
        guess_age=guess_df_age.median()
        age_guess[i,j]=int(guess_age)

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

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

Out[25]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_band	Pedigree_I
0	6	148.0	1	0	1	0	50	1	(22.367, 44.733]	(0.0757, 0
1	1	NaN	1	0	1	0	31	0	(22.367, 44.733]	(0.0757, 0
2	8	183.0	1	0	1	0	29	1	(22.367, 44.733]	(0.0757, 0
3	1	NaN	1	0	1	0	21	0	(22.367, 44.733]	(0.0757, 0
4	0	137.0	0	0	1	2	33	1	(22.367, 44.733]	(1.639,

In [26]:
```
glucose_guess=np.zeros((3,3),dtype=int)

for i in range(0,3):
    for j in range(0,3):
        guess_df_glucose=data[(data['Pedigree']==i) & (data['BP']==j)]['gluc
        guess_glucose=guess_df_glucose.median()
        glucose_guess[i,j]=int(guess_glucose)

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

data.glucoseLevel=data.glucoseLevel.astype(int)

data.head()
```

Out[26]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_band	Pedigree_l
0	6	148	1	0	1	0	50	1	(22.367, 44.733]	(0.0757, 0
1	1	112	1	0	1	0	31	0	(22.367, 44.733]	(0.0757, 0
2	8	183	1	0	1	0	29	1	(22.367, 44.733]	(0.0757, 0
3	1	112	1	0	1	0	21	0	(22.367, 44.733]	(0.0757, 0
4	0	137	0	0	1	2	33	1	(22.367, 44.733]	(1.639,

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

In [27]:
```
# YOUR CODE HERE
#raise NotImplementedError()

data['glucoseLevel']=pd.cut(data['glucoseLevel'],4,labels=[0,1,2,3])

data['Age']=pd.cut(data['Age'],4,labels=[0,1,2,3])

data.head()
```

Out[27]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_band	Pedigree_l
0	6	2	1	0	1	0	1	1	(22.367, 44.733]	(0.0757, 0
1	1	2	1	0	1	0	0	0	(22.367, 44.733]	(0.0757, 0
2	8	3	1	0	1	0	0	1	(22.367, 44.733]	(0.0757, 0
3	1	2	1	0	1	0	0	0	(22.367, 44.733]	(0.0757, 0
4	0	2	0	0	1	2	0	1	(22.367, 44.733]	(1.639,

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 [37]:
```
train_df, test_df = train_df, test_df = train_test_split(data,test_size=0.1
X_train = train_df.iloc[:,0:7]
Y_train = train_df.iloc[:,7]
X_test  = test_df.iloc[:,0:7]
Y_test = test_df.iloc[:,7]

X_train.shape, Y_train.shape, X_test.shape
```

Out[37]:
```
((652, 7), (652,), (116, 7))
```

In [38]:
```
# 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=  0.7254601226993865
logreg test accuracy=  0.8103448275862069
```

In [39]:
```
# 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=  0.691717791411043
perceptron test accuracy=  0.6724137931034483

/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/stochastic_gr
adient.py:128: FutureWarning: max_iter and tol parameters have been added
in <class 'sklearn.linear_model.perceptron.Perceptron'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not No
ne, 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 [40]:
```
# 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=  0.8650306748466258
random_forest test accuracy=  0.6982758620689655
```