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 [71]:
```
import numpy as np
import pandas as pd
```

In [72]:
```
 # 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
```

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 [73]:
```
#Read data & print it
data = pd.read_csv("diabetesdata.csv")
data.head(5)
```

Out[73]:

	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 [74]:
```
NullsPerColumn = (data.isnull().sum()/len(data))
NullsPerColumn
```

Out[74]:
```
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
```

In [75]:
```
###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 [76]:
```
PercentNull = sum([True for idx,row in data.iterrows() if any(row.isnull())])/len(data)
PercentNull
```

Out[76]:
```
0.08333333333333333
```

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

In [77]:
```
#split values
from sklearn.model_selection import train_test_split
train_df, test_df = train_test_split(data,test_size=0.15)
print ('Number of samples in training data:',len(train_df))
print ('Number of samples in test data:',len(test_df))
```

```
Number of samples in training data: 652
Number of samples in test data: 116
```

In [78]:
```
###RUN THIS CELL BUT DO NOT ALTER IT
np.testing.assert_almost_equal(float(len(train_df))/float(len(data)), 0.8489583333333334, 1)
np.testing.assert_almost_equal(float(len(test_df))/float(len(data)), 0.15104166666666666, 1)
```

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

In [79]:
```
train_df = train_df.fillna(train_df.mean())
test_df = test_df.fillna(test_df.mean())
```

In [80]:
```
###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 [81]:
```
X_train = train_df[['TimesPregnant','glucoseLevel','BP','insulin','BMI','Pedigree','Age']]
Y_train = train_df['IsDiabetic']
X_test  = test_df[['TimesPregnant','glucoseLevel','BP','insulin','BMI','Pedigree','Age']]
Y_test = test_df['IsDiabetic']
```

In [82]:
```
###RUN THIS CELL BUT DO NOT ALTER IT
assert [X_train.shape, Y_train.shape, X_test.shape,Y_test.shape] == [(652, 7), (652,), (116, 7),(116,)]
```

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

In [83]:
```
# Logistic Regression

logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
logreg_train_acc = sum(logreg.predict(X_train) == Y_train)/len(Y_train)
logreg_test_acc = sum(logreg.predict(X_test) == Y_test)/len(Y_test)
print ('logreg training acuracy= ',logreg_train_acc)
print('logreg test accuracy= ',logreg_test_acc)
```

```
logreg training acuracy=  0.7699386503067485
logreg test accuracy=  0.7931034482758621
```

In [84]:
```
# 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=  0.6733128834355828
perceptron test accuracy=  0.6724137931034483
```

```
/anaconda3/envs/data-x/lib/python3.6/site-packages/sklearn/linear_model/stochastic_gradient.py:128: Fut
ureWarning: max_iter and tol parameters have been added in <class 'sklearn.linear_model.perceptron.Perc
eptron'> in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not None,
max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1
e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
```

In [85]:
```
# 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=  0.8021472392638037
adaboost test accuracy=  0.8017241379310345
```

In [86]:
```
# 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=  0.8103448275862069
```

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 really because std for glucoselevel and age are pretty much high so I wouldn't use mean imputation. Normally, there are median imputation/ Regression imputation. Or I would like to select values randomly in glucoselevel to replace the Nan values in glucouselevel.

In [87]:
```
data.describe()
```

Out[87]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
count	768.000000	734.000000	768.000000	768.000000	768.000000	768.000000	735.000000	768.000000
mean	3.845052	121.016349	69.105469	79.799479	31.992578	0.471876	33.353741	0.348958
std	3.369578	31.660240	19.355807	115.244002	7.884160	0.331329	11.772944	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	141.000000	80.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	846.000000	67.100000	2.420000	81.000000	1.000000

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 [88]:
```
# YOUR CODE HERE
#raise NotImplementedError()
data['BMI_band'] = pd.cut(data['BMI'], 3)
data['Pedigree_band'] = pd.cut(data['Pedigree'], 3)
data.head(5)
```

Out[88]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_band	Pedigree_band
0	6	148.0	72	0	33.6	0.627	50.0	1	(22.367, 44.733]	(0.0757, 0.859]
1	1	NaN	66	0	26.6	0.351	31.0	0	(22.367, 44.733]	(0.0757, 0.859]
2	8	183.0	64	0	23.3	0.672	NaN	1	(22.367, 44.733]	(0.0757, 0.859]
3	1	NaN	66	94	28.1	0.167	21.0	0	(22.367, 44.733]	(0.0757, 0.859]
4	0	137.0	40	168	43.1	2.288	33.0	1	(22.367, 44.733]	(1.639, 2.42]

1a. Print the category intervals for `BMI_band` & `Pedigree_band`.

In [89]:
```
print('BMI_Band_Interval: ' + str(pd.unique(data['BMI_band'])))
```

```
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]]
```

In [90]:
```
print('Pedigree_Band_Interval: ' + str(pd.unique(data['Pedigree_band'])))
```

```
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 [91]:
```
# YOUR CODE HERE
#raise NotImplementedError()

pedigree_DiabeticRatio = data.groupby(('Pedigree_band'), as_index=False).mean()
pedigree_DiabeticRatio
```

Out[91]:

	Pedigree_band	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
0	(0.0757, 0.859]	3.870073	120.191424	68.757664	75.702190	31.659562	0.384975	33.307339	0.327007
1	(0.859, 1.639]	3.932432	125.500000	72.486486	105.878378	34.739189	1.090770	34.375000	0.540541
2	(1.639, 2.42]	1.222222	145.000000	67.777778	177.222222	34.755556	1.997333	28.555556	0.444444

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

In [92]:
```
# YOUR CODE HERE
#raise NotImplementedError()

BMI_DiabeticRatio =  data.groupby(('BMI_band'), as_index=False).mean()
BMI_DiabeticRatio
```

Out[92]:

	BMI_band	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
0	(-0.0671, 22.367]	2.568627	102.297872	54.803922	36.823529	16.194118	0.380255	30.591837	0.039216
1	(22.367, 44.733]	3.964758	121.767228	69.566814	81.449339	32.284875	0.475261	33.537634	0.358297
2	(44.733, 67.1]	3.388889	132.470588	80.638889	109.472222	48.844444	0.537639	33.800000	0.611111

In [93]:
```
###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 [94]:
```
# 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])
```

In [95]:
```
###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 [96]:
```
# YOUR CODE HERE
#raise NotImplementedError()
```

In [97]:
```
guess_ages = np.zeros((3,3),dtype=int) #initialize
guess_ages
```

Out[97]:
```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

In [98]:
```
# Fill the NA's for the Age columns
# with "qualified guesses"
for i in range(0, 3):
    for j in range(0,3):
        guess_df = data[(data['BMI'] == i) \
                        &(data['BP'] == j)]['Age'].dropna()

        # Extract the median age for this group
        # (less sensitive) to outliers
        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)
print ('\nAssigning age values to NAN age values in the dataset...')

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


data['Age'] = data['Age'].astype(int)
print()
print('Done! \n\n\n')
data.head()
```

```
Guess_Age table:
 [[24 25 55]
 [29 29 37]
 [33 32 31]]

Assigning age values to NAN age values in the dataset...

Done!
```

Out[98]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_band	Pedigree_band
0	6	148.0	1	0	1	0	50	1	(22.367, 44.733]	(0.0757, 0.859]
1	1	NaN	1	0	1	0	31	0	(22.367, 44.733]	(0.0757, 0.859]
2	8	183.0	1	0	1	0	29	1	(22.367, 44.733]	(0.0757, 0.859]
3	1	NaN	1	0	1	0	21	0	(22.367, 44.733]	(0.0757, 0.859]
4	0	137.0	0	0	1	2	33	1	(22.367, 44.733]	(1.639, 2.42]

In [99]:
```
guess_glucoseLevel = np.zeros((3,3),dtype=int) #initialize
guess_glucoseLevel
```

Out[99]:
```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

In [100]:
```
# Fill the NA's for the glucoseLevel columns
# with "qualified guesses"
for i in range(0, 3):
    for j in range(0,3):
        guess_df = data[(data['BP'] == i) \
                        &(data['Pedigree'] == j)]['glucoseLevel'].dropna()

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

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


print('Guess_glucoseLevel table:\n',guess_glucoseLevel)
print ('\nAssigning age values to NAN age values in the dataset...')

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)
print()
print('Done! \n\n\n')
data.head()
```

```
Guess_glucoseLevel table:
 [[115 127 137]
 [112 115 149]
 [133 129 159]]

Assigning age values to NAN age values in the dataset...

Done!
```

Out[100]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_band	Pedigree_band
0	6	148	1	0	1	0	50	1	(22.367, 44.733]	(0.0757, 0.859]
1	1	127	1	0	1	0	31	0	(22.367, 44.733]	(0.0757, 0.859]
2	8	183	1	0	1	0	29	1	(22.367, 44.733]	(0.0757, 0.859]
3	1	127	1	0	1	0	21	0	(22.367, 44.733]	(0.0757, 0.859]
4	0	137	0	0	1	2	33	1	(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 [101]:
```
# 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(5)
```

Out[101]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_band	Pedigree_band
0	6	2	1	0	1	0	1	1	(22.367, 44.733]	(0.0757, 0.859]
1	1	2	1	0	1	0	0	0	(22.367, 44.733]	(0.0757, 0.859]
2	8	3	1	0	1	0	0	1	(22.367, 44.733]	(0.0757, 0.859]
3	1	2	1	0	1	0	0	0	(22.367, 44.733]	(0.0757, 0.859]
4	0	2	0	0	1	2	0	1	(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 [102]:
```
train_df, test_df = train_test_split(data,test_size=0.15,random_state=100)
X_train = train_df[['TimesPregnant','glucoseLevel','BP','insulin','BMI','Pedigree','Age']]
Y_train = train_df['IsDiabetic']
X_test  = test_df[['TimesPregnant','glucoseLevel','BP','insulin','BMI','Pedigree','Age']]
Y_test= test_df['IsDiabetic']
X_train.shape, Y_train.shape, X_test.shape
```

Out[102]:
```
((652, 7), (652,), (116, 7))
```

In [103]:
```
# Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
logreg_train_acc = sum(logreg.predict(X_train) == Y_train)/len(Y_train)
logreg_test_acc = sum(logreg.predict(X_test) == Y_test)/len(Y_test)
print ('logreg training acuracy= ',logreg_train_acc)
print('logreg test accuracy= ',logreg_test_acc)
```

```
logreg training acuracy=  0.754601226993865
logreg test accuracy=  0.7155172413793104
```

In [104]:
```
# 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=  0.6641104294478528
perceptron test accuracy=  0.646551724137931
```

```
/anaconda3/envs/data-x/lib/python3.6/site-packages/sklearn/linear_model/stochastic_gradient.py:128: Fut
ureWarning: max_iter and tol parameters have been added in <class 'sklearn.linear_model.perceptron.Perc
eptron'> in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not None,
max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1
e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
```

In [105]:
```
# 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=  0.8788343558282209
random_forest test accuracy=  0.6379310344827587
```