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
```

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
```

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
5	5	116.0	74	0	25.6	0.201	30.0	0
6	3	78.0	50	88	31.0	0.248	26.0	1
7	10	115.0	0	0	35.3	0.134	29.0	0
8	2	197.0	70	543	30.5	0.158	53.0	1
9	8	NaN	96	0	0.0	0.232	54.0	1
10	4	110.0	92	0	37.6	0.191	NaN	0
11	10	168.0	74	0	38.0	0.537	34.0	1
12	10	139.0	80	0	27.1	1.441	57.0	0
13	1	NaN	60	846	30.1	0.398	59.0	1
14	5	166.0	72	175	25.8	0.587	51.0	1
15	7	100.0	0	0	30.0	0.484	32.0	1
16	0	NaN	84	230	45.8	0.551	31.0	1
17	7	107.0	74	0	29.6	0.254	31.0	1
18	1	103.0	30	83	43.3	0.183	33.0	0
19	1	115.0	70	96	34.6	0.529	32.0	1
20	3	126.0	88	235	39.3	0.704	27.0	0
21	8	99.0	84	0	35.4	0.388	50.0	0
22	7	196.0	90	0	39.8	0.451	41.0	1
23	9	119.0	80	0	29.0	0.263	29.0	1
24	11	143.0	94	146	36.6	0.254	51.0	1
25	10	125.0	70	115	31.1	0.205	41.0	1
26	7	147.0	76	0	39.4	0.257	43.0	1
27	1	97.0	66	140	23.2	0.487	NaN	0
28	13	NaN	82	110	22.2	0.245	57.0	0
29	5	117.0	92	0	34.1	0.337	38.0	0
...	...	...	...	...	...	...	...	...
738	2	99.0	60	160	36.6	0.453	NaN	0

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
739	1	102.0	74	0	39.5	0.293	42.0	1
740	11	120.0	80	150	42.3	0.785	48.0	1
741	3	102.0	44	94	30.8	0.400	26.0	0
742	1	109.0	58	116	28.5	0.219	22.0	0
743	9	140.0	94	0	32.7	0.734	45.0	1
744	13	153.0	88	140	40.6	1.174	39.0	0
745	12	100.0	84	105	30.0	0.488	46.0	0
746	1	147.0	94	0	49.3	0.358	27.0	1
747	1	81.0	74	57	46.3	1.096	32.0	0
748	3	187.0	70	200	36.4	0.408	36.0	1
749	6	162.0	62	0	24.3	0.178	50.0	1
750	4	136.0	70	0	31.2	1.182	22.0	1
751	1	121.0	78	74	39.0	0.261	28.0	0
752	3	108.0	62	0	26.0	0.223	25.0	0
753	0	181.0	88	510	43.3	0.222	26.0	1
754	8	154.0	78	0	32.4	0.443	45.0	1
755	1	128.0	88	110	36.5	1.057	37.0	1
756	7	137.0	90	0	32.0	0.391	39.0	0
757	0	123.0	72	0	36.3	0.258	52.0	1
758	1	106.0	76	0	37.5	0.197	26.0	0
759	6	190.0	92	0	35.5	0.278	66.0	1
760	2	88.0	58	16	28.4	0.766	22.0	0
761	9	170.0	74	0	44.0	0.403	43.0	1
762	9	89.0	62	0	22.5	0.142	33.0	0
763	10	101.0	76	180	32.9	0.171	63.0	0
764	2	122.0	70	0	36.8	0.340	27.0	0
765	5	121.0	72	112	26.2	0.245	30.0	0
766	1	126.0	60	0	30.1	0.349	47.0	1
767	1	93.0	70	0	30.4	0.315	23.0	0

768 rows × 8 columns

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

In [4]:

```
NullsPerColumn = data.isnull().sum()/data.shape[0]
NullsPerColumn = pd.DataFrame(NullsPerColumn)
NullsPerColumn.columns = ['Percentage Null']
print(NullsPerColumn)
```

```
               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 =  1-data.dropna().shape[0]/data.shape[0]
print('PercentNull=%f' %PercentNull)
```

```
PercentNull=0.083333
```

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)), 0.848958333333
np.testing.assert_almost_equal(float(len(test_df))/float(len(data)), 0.151041666666
```

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

In [9]:

```
train_df = train_df.fillna(0)+train_df.isna()*train_df.mean()
test_df = test_df.fillna(0)+test_df.isna()*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
Y_train=Y_train.astype('int')
Y_test=Y_test.astype('int')

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
import warnings
warnings.filterwarnings('ignore')

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=  0.36503067484662577
perceptron test accuracy=  0.3103448275862069
```

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=  0.8174846625766872
adaboost test accuracy=  0.7672413793103449
```

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=  0.9861963190184049
random_forest test accuracy=  0.7327586206896551
```

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?

No. Mean imputation ignores the relationship between variables and may lead to outliers when fitted in a regression model. Other ways to impute the data include regression imputation which is obta Hot deck imputation

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
# raise NotImplementedError()
data['BMI_band'] = pd.cut(data['BMI'], bins=3)
data['Pedigree_band'] = pd.cut(data['Pedigree'], bins=3)
data.head(5)
```

Out[17]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_band	Pedigr
0	6	148.0	72	0	33.6	0.627	50.0	1	(22.367, 44.733]	(0.075
1	1	NaN	66	0	26.6	0.351	31.0	0	(22.367, 44.733]	(0.075
2	8	183.0	64	0	23.3	0.672	NaN	1	(22.367, 44.733]	(0.075
3	1	NaN	66	94	28.1	0.167	21.0	0	(22.367, 44.733]	(0.075
4	0	137.0	40	168	43.1	2.288	33.0	1	(22.367, 44.733]	(1.6

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

In [18]:

```
print('BMI_Band_Interval: ' + str(data['BMI_band'].cat.categories))
print('Pedigree_Band_Interval: ' + str(data['Pedigree_band'].cat.categories))
```

```
BMI_Band_Interval: IntervalIndex([(-0.0671, 22.367], (22.367, 44.733],
(44.733, 67.1]]
              closed='right',
              dtype='interval[float64]')
Pedigree_Band_Interval: IntervalIndex([(0.0757, 0.859], (0.859, 1.63
9], (1.639, 2.42]]
              closed='right',
              dtype='interval[float64]')
```

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

In [19]:

```
# YOUR CODE HERE
#raise NotImplementedError()

pedigree_DiabeticRatio = data.groupby(['Pedigree_band'],as_index=False)['IsDiabetic
pedigree_DiabeticRatio
```

Out[19]:

	Pedigree_band	IsDiabetic
0	(0.0757, 0.859]	0.327007
1	(0.859, 1.639]	0.540541
2	(1.639, 2.42]	0.444444

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

In [20]:

```
# YOUR CODE HERE
# raise NotImplementedError()

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

Out[20]:

	BMI_band	IsDiabetic
0	(-0.0671, 22.367]	0.039216
1	(22.367, 44.733]	0.358297
2	(44.733, 67.1]	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
```

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]:

```
# YOUR CODE HERE
# raise NotImplementedError()
data['BP'] = pd.cut(data['BP'], bins=3, labels=[0,1,2])
data['insulin'] = pd.cut(data['insulin'], bins=3, labels=[0,1,2])
data['BMI'] = pd.cut(data['BMI'], bins=3, labels=[0,1,2])
data['Pedigree'] = pd.cut(data['Pedigree'], bins=3, labels=[0,1,2])
```

In [23]:

```
###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]:

```
# YOUR CODE HERE
# raise NotImplementedError()
guess_age = 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()
            guess_age[i,j] = int(age_guess)

guess_glucoseLevel = np.zeros((3,3), dtype=int)
for i in range(0, 3):
        for j in range(0,3):
            guess_df = data[(data['Pedigree'] == i) \
                            &(data['BP'] == j)]['glucoseLevel'].dropna()
            glucose_guess = guess_df.median()
            guess_glucoseLevel[i,j] = int(glucose_guess)

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_age[i,j]
            data.loc[ (data.glucoseLevel.isnull()) & (data.Pedigree == i) \
                    & (data.BP == j),'glucoseLevel'] = guess_glucoseLevel[i,j]


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

In [25]:

```
guess_age
```

Out[25]:

```
array([[24, 29, 33],
       [25, 29, 32],
       [55, 37, 31]])
```

In [26]:

```
guess_glucoseLevel
```

Out[26]:

```
array([[115, 112, 133],
       [127, 115, 129],
       [137, 149, 159]])
```

In [27]:

```
data.head()
```

Out[27]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_band	Pedigr
0	6	148	1	0	1	0	50	1	(22.367, 44.733]	(0.075
1	1	112	1	0	1	0	31	0	(22.367, 44.733]	(0.075
2	8	183	1	0	1	0	29	1	(22.367, 44.733]	(0.075
3	1	112	1	0	1	0	21	0	(22.367, 44.733]	(0.075
4	0	137	0	0	1	2	33	1	(22.367, 44.733]	(1.6

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

In [28]:

```
# YOUR CODE HERE
# raise NotImplementedError()
data['Age'] = pd.cut(data['Age'], bins=4, labels=[0,1,2,3])
data['glucoseLevel'] = pd.cut(data['glucoseLevel'], bins=4, labels=[0,1,2,3])
data.head()
```

Out[28]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_band	Pedigr
0	6	2	1	0	1	0	1	1	(22.367, 44.733]	(0.075
1	1	2	1	0	1	0	0	0	(22.367, 44.733]	(0.075
2	8	3	1	0	1	0	0	1	(22.367, 44.733]	(0.075
3	1	2	1	0	1	0	0	0	(22.367, 44.733]	(0.075
4	0	2	0	0	1	2	0	1	(22.367, 44.733]	(1.6

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 [29]:

```
train_df, test_df = train_test_split(data, test_size=0.15)
X_train = train_df.drop(["IsDiabetic",'BMI_band','Pedigree_band'], axis=1)
Y_train = train_df["IsDiabetic"]
X_test  = test_df.drop(["IsDiabetic",'BMI_band','Pedigree_band'], axis=1)
Y_test= test_df["IsDiabetic"]
X_train.shape, Y_train.shape, X_test.shape
```

Out[29]:

```
((652, 7), (652,), (116, 7))
```

In [30]:

```
# 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=  0.754601226993865
logreg test accuracy=  0.6724137931034483
```

In [31]:

```
# Perceptron

import warnings
warnings.filterwarnings('ignore')

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=  0.5862068965517241
```

In [32]:

```
# 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=  0.6551724137931034
```