Data-X Spring 2019: Homework 06

Name : McClain Thiel

SID : 3034003600

Course (IEOR 135/290) : IEOR 135

Machine Learning

In this homework, you will do some exercises with prediction. We will cover these algorithms in class, but this is for you to have some hands on with these in scikit-learn. You can refer - https://github.com/ikhlaqsidhu/data-x/blob/master/05a-tools-predicition-titanic/titanic.ipynb (https://github.com/ikhlaqsidhu/data-x/blob/master/05a-tools-predicition-titanic/titanic.ipynb)

Display all your outputs.

import numpy as np import pandas as pd

In [2]:
```
 # machine learning libraries
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import Perceptron
from sklearn.tree import DecisionTreeClassifier

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')
```

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 the head
df = pd.read_csv('diabetesdata.csv')
df.head()
```

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 Null values in each column and display it.

In [4]:
```
percent_missing = df.isnull().sum() * 100 / len(df)
```

3. Split `data` into `train_df` and `test_df` with 15% as test.

In [5]:
```
from sklearn.model_selection import train_test_split

train_df, test_df = train_test_split(df, test_size = .15)
both = [train_df, test_df]
```

4. Display the means of the features in train and test sets. Replace the null values in `train_df` and `test_df` with the mean of EACH feature column separately for train and test. Display head of the dataframes.

In [6]:
```
print('Train: \n ', both[0].mean(), ' \n Test: \n', both[1].mean())
both[0] = both[0].fillna(both[0].mean())
both[1] = both[1].fillna(both[1].mean())
display(both[0].head(), both[1].head())
```

```
Train:
  TimesPregnant      3.840491
glucoseLevel     121.932800
BP                68.946319
insulin           80.476994
BMI               31.965491
Pedigree           0.472489
Age               33.430177
IsDiabetic         0.357362
dtype: float64
 Test:
 TimesPregnant      3.870690
glucoseLevel     115.761468
BP                70.000000
insulin           75.991379
BMI               32.144828
Pedigree           0.468431
Age               32.928571
IsDiabetic         0.301724
dtype: float64
```

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
544	1	88.0	78	76	32.0	0.365	29.0	0
762	9	89.0	62	0	22.5	0.142	33.0	0
580	0	151.0	90	0	42.1	0.371	21.0	1
116	5	124.0	74	0	34.0	0.220	38.0	1
214	9	112.0	82	175	34.2	0.260	36.0	1

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
344	8	95.0	72	0	36.8	0.485	57.0	0
730	3	130.0	78	79	28.4	0.323	34.0	1
639	1	100.0	74	46	19.5	0.149	28.0	0
487	0	173.0	78	265	46.5	1.159	58.0	0
37	9	102.0	76	0	32.9	0.665	46.0	1

5. 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 [7]:
```
X_train, X_test = both[0].drop('IsDiabetic', axis = 1), both[1].drop('Is
Diabetic', axis = 1)
Y_train, Y_test = both[0]['IsDiabetic'], both[1]['IsDiabetic']
```

6. Use this dataset to train perceptron, logistic regression and random forest models using 15% test split. Report training and test accuracies. Try different hyperparameter values for these models and see if you can improve your accuracies.

In [8]:
```
# 6a. Logistic Regression
log = LogisticRegression()
log.fit(X_train, Y_train)
print('Initial accuracy: ', log.score(X_test, Y_test) * 100 , '%')
print('Train accuracy: ', log.score(X_train, Y_train) * 100 , '%')

#hyperparameter tuning
from sklearn.model_selection import GridSearchCV

grid={"C":np.logspace(-3,3,7), "penalty":["l1","l2"]}# l1 lasso l2 ridge
logreg=LogisticRegression()
logreg_cv=GridSearchCV(logreg,grid,cv=10)
logreg_cv.fit(X_train,Y_train)

print("Tuned accuracy 1 :",logreg_cv.best_score_* 100 , '%')
#trying again with normalized inputs

def normalize(z):
    return (z-np.min(z)/(np.max(z) - np.min(z)))
X2_train, X2_test = normalize(X_train), normalize(X_test)

grid2={"C":[.001, .01, .1, 1, 10, 100, 1000], "penalty":["l1","l2"]}
logreg2=LogisticRegression()
logreg_cv2=GridSearchCV(logreg2, grid2)
logreg_cv2.fit(X2_train, Y_train)
print("Tuned accuracy 2 :",logreg_cv2.best_score_* 100 , '%')

#maybe try k-fold
```

```
Initial accuracy:  74.13793103448276 %
Train accuracy:  77.76073619631902 %
Tuned accuracy 1 : 78.06748466257669 %
Tuned accuracy 2 : 77.30061349693251 %
```

In [9]:
```
# 6b. Perceptron
perceptron = Perceptron()                                        # instantia
te
perceptron.fit(X_train, Y_train)                                 # fit
acc_perceptron = perceptron.score(X_test, Y_test)                  # predict
+ evalaute

print('Perceptron labeling accuracy:', str(round(acc_perceptron*100,2)),
'%')
```

```
Perceptron labeling accuracy: 52.59 %
```

In [10]:
```
# 6c. Random Forest
# Random Forest
random_forest = RandomForestClassifier(n_estimators=430)   # instantiate
random_forest.fit(X_train, Y_train)                         # fit
acc_rf = random_forest.score(X_test, Y_test)                 # predict
 + evaluate

print('Rndom forest labeling accuracy:', str(round(acc_rf*100,2)),'%')
```

```
Rndom forest labeling accuracy: 72.41 %
```

7. For your logistic regression model -

a . Compute the log probability of classes in `IsDiabetic` for the first 10 samples of your train set and display it. Also display the predicted class for those samples from your logistic regression model trained before.

In [17]:
```
top10 = X_train[0:][:10]
print('The probability that the person is diabetic based on out model in
log space is: \n', log.predict_log_proba(top10), '\nFormat: [p(isNotDiab
etic, p(isDiabetic)]')

print('Predictions from model: ' , log.predict(top10))

confidence = np.mean([max(x) for x in log.predict_proba(top10)])
confidence
```

```
The probability that the person is diabetic based on out model in log s
pace is:
 [[-0.10516489 -2.30434746]
 [-0.20838814 -1.67073821]
 [-0.5560524  -0.85206888]
 [-0.47316632 -0.97558024]
 [-0.49480879 -0.94080751]
 [-0.1333229  -2.08090221]
 [-0.13733816 -2.05319239]
 [-0.43947998 -1.03386841]
 [-0.24128063 -1.54001038]
 [-0.75555983 -0.63440209]]
Format: [p(isNotDiabetic, p(isDiabetic)]
Predictions from model:  [0 0 0 0 0 0 0 0 0 1]
```

Out[17]:
```
0.7225353117073564
```

b . Now compute the log probability of classes in `IsDiabetic` for the first 10 samples of your test set and display it. Also display the predicted class for those samples from your logistic regression model trained before. (using the model trained on the training set)

In [18]:
```
top10test = X_test[0:][:10]
print('The probability that the person is diabetic based on out model in log space is: \n', log.predict_log_proba(top10test), '\nFormat: [p(isNot Diabetic, p(isDiabetic)]')

print('Predictions from model: ' , log.predict(top10test))
test_confidence = np.mean([max(x) for x in log.predict_proba(top10test
)])
test_confidence
```

```
The probability that the person is diabetic based on out model in log s
pace is:
 [[-0.54543642 -0.86652196]
 [-0.29213142 -1.37306387]
 [-0.06047465 -2.83561595]
 [-1.59725275 -0.22621318]
 [-0.54931179 -0.86120379]
 [-0.21447441 -1.64488614]
 [-1.83797721 -0.17332892]
 [-0.30462007 -1.33713657]
 [-0.48759922 -0.95217426]
 [-0.07066733 -2.68489752]]
Format: [p(isNotDiabetic, p(isDiabetic)]
Predictions from model:  [0 0 0 1 0 0 1 0 0 0]
```

Out[18]:
```
0.7573572569531293
```

c . What can you interpret from the log probabilities and the predicted classes?

I generally don't opperate in log space for probabilities, but the data basically indicates how sure the model is of each one of its predictions. If the model reports isDiabetic == 0 with probability of log(.94) or something similar, I would be more inclined to belive it that if it reported a confidence of log(.53) or something similar. This particular set of training data seems to suggest that the model his very conifident in most of it's predictions on the training data which makes sense because its seen this data before. What is suprising is that the model has a higher average conficence for the test data.

8. Is mean imputation is the best type of imputation (as we did in 4.) to use? Why or why not? What are some other ways to impute the data?

It seems pretty accurate. 80% accuracy isn't terrible but it could still be better. Some other methods include MICE, SVD, KNN, FKM and bPCA. We could also just ignore any row that has missing data. I'd argue mean imputaion was the best to use in this situation because it was easy to use, produced good results and the data was such that the mean was a reasonable value. If the data was highly polarized and produced a mean that was far from all other values this method may note have been as effective.

Extra Credit (2 pts) - MANDATORY for students enrolled in IEOR 290

9. Implement the K-Nearest Neighbours (https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm (https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm)) algorithm for k=1 from scratch in python (do not use KNN from existing libraries). KNN uses Euclidean distance to find nearest neighbors. Split your dataset into test and train as before. Also fill in the null values with mean of features as done earlier. Use this algorithm to predict values for 'IsDiabetic' for your test set. Display your accuracy.

In [204]:

```
import operator

def euclideanDistance(data1, data2, length):
    distance = 0
    for x in range(length):
        distance += np.square(data1[x] - data2[x])
    return np.sqrt(distance)

def getNeig(train, test, k=1):
    distances = []
    length = len(test) -1
    for x in range(len(train)):
        dist = euclideanDistance(test, train.iloc[x], length)
        distances.append((train.iloc[x], dist))

    distances.sort(key= operator.itemgetter(1))
    neig = []
    for x in range(k):
        neig.append(distances[x][0])
    return neig

def getResponse(neig):
    classVotes = {}
    for x in range(len(neig)):
        resp = neig[x][-1]
        if resp in classVotes:
            classVotes[resp] += 1
        else:
            classVotes[resp] = 1
    sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(0),
reverse=True)
    return sortedVotes[0][0]


correct = 0
for x in range(X_test.shape[0]):
    prediction = getResponse(getNeig(train_df, X_test.iloc[x]))
    if prediction == Y_test.iloc[x]:
        correct+=1

percent_correct = correct/X_test.shape[0]

print('Accuracy of KNN: ', percent_correct*100, '%')
#alarmingly slow
```

```
Accuracy of KNN:  71.55172413793103 %
```

In [ ]: