Naïve Bayes Classifier for Diabetes Diagnosis
Jericho McLeod
CSI-777

Step one with python is always importing the necessary libraries. I will be using Pandas for the dataframe data structure, random for my k-fold cross validation, and the naive-bayes classifier from Scikit-Learn. Scikit-Learn offers a k-fold cross validation function, but it is relatively simple to implement manually.

In [1]:
```
import numpy as np
import pandas as pd
import random
import sklearn.naive_bayes
k = 10
```

The next step is importing data. Since I am using Pandas dataframes I will use the pandas function for CSV import. This is also more concise than using the csv library. At this point I will also convert the outcomes to binary, such that:
'Diabetes' == 1
'No Diabetes' == 0

In [2]:
```
df = pd.read_csv('./Diabetes Dataset/diabetes.csv', skiprows=0)
bin_diabetes = []
for i in df['Diabetes']:
    if i == 'tested_positive':
        bin_diabetes.append(1)
    else:
        bin_diabetes.append(0)
del df['Diabetes']
df['Diabetes_num'] = bin_diabetes

df.head()
```

Out[2]:

	timesPreg	Glucose	blood pressure	triceps skin fold thickness	2hr-serum-insulin	BMI	Diabetes pedigree fnc	age	Diabetes_num
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

The data has value ranges that are very small and very large, so a transformation is necessary. I will use a z-transformation, as it is relatively simple to implement and typically gives relatively good results.

Note that running this cell repeatedly will cause the '_z' character to be appended to the column headers; if this occurs, re-run the prior cell to re-import the data.

In [3]:
```
cols = list(df.columns)
cols.remove('Diabetes_num')
cols_z = []
for i in cols:
    col_zscore = i + '_z'
    cols_z.append(col_zscore)
    df[col_zscore] = (df[i] - df[i].mean())/df[i].std(ddof=1)
    del df[i]
df.head()
```

Out[3]:

	Diabetes_num	timesPreg_z	Glucose_z	blood pressure_z	triceps skin fold thickness_z	2hr-serum-insulin_z	BMI_z	Diabet pedigr fnc
0	1	0.639530	0.847771	0.149543	0.906679	-0.692439	0.203880	0.4681
1	0	-0.844335	-1.122665	-0.160441	0.530556	-0.692439	-0.683976	-0.3648
2	1	1.233077	1.942458	-0.263769	-1.287373	-0.692439	-1.102537	0.6040
3	0	-0.844335	-0.997558	-0.160441	0.154433	0.123221	-0.493721	-0.9201
4	1	-1.141108	0.503727	-1.503707	0.906679	0.765337	1.408828	5.4813

The following cell appends a column of random values from 0 to 1: this is used for the k-fold validation subsetting.

In [4]:
```
rand_vals = []
for i in range(len(df)):
    rand_vals.append(random.random())
df['random']=rand_vals
df.head()
```

Out[4]:

	Diabetes_num	timesPreg_z	Glucose_z	blood pressure_z	triceps skin fold thickness_z	2hr-serum-insulin_z	BMI_z	Diabet pedigr fnc
0	1	0.639530	0.847771	0.149543	0.906679	-0.692439	0.203880	0.4681
1	0	-0.844335	-1.122665	-0.160441	0.530556	-0.692439	-0.683976	-0.3648
2	1	1.233077	1.942458	-0.263769	-1.287373	-0.692439	-1.102537	0.6040
3	0	-0.844335	-0.997558	-0.160441	0.154433	0.123221	-0.493721	-0.9201
4	1	-1.141108	0.503727	-1.503707	0.906679	0.765337	1.408828	5.4813

The following cell essentially does everything of importance in this model at once.

1. Determine the min and max 'random' values
2. Use these values to subset training and testing datasets
3. Instantiate and fit the Naïve Bayes model with the training data
4. Iterate through the testing data and compare classified values with actual values
5. Determine the accuracy within the fold
6. Add the value to an overall accuracy variable

Then, when the loop is completed, it divides the overall accuracy value by K and prints the result.

In [5]:
```
overall =0
for i in range(k):
    rand_min,rand_max = i/k,(i+1)/k
    df_test  = df.loc[(df['random']>=rand_min) & (df['random']< rand_max
)]
    df_train = df.loc[(df['random']< rand_min) | (df['random']>=rand_max
)]
    model = sklearn.naive_bayes.GaussianNB()
    features = df_train[cols_z]
    labels = df_train['Diabetes_num']
    model.fit(features,labels)
    correct = 0
    for index,row in df_test.iterrows():
        row_data = []
        for i in row:
            row_data.append(i)
        actual = row_data.pop(0)
        del row_data[-1]
        classified = model.predict([row_data])
        if actual - classified[0]  ==0:
            correct+=1
    overall+=correct/len(df_test['Diabetes_num'])
overall=overall/k
print('Overall Accuracy with K='+str(k)+' fold validation:',overall)
```

```
Overall Accuracy with K=10 fold validation: 0.7516753232216782
```

It is my opinion that 74% is garbage. This is due in no small part to the fact that I peeked at the data before starting and noticed missing data. Tricep skin fold measurements are an indicator of BMI, so I am not as concerned with these beyond perhaps eliminating the measurement for lack of independence. However, blood pressure or 2-hr serum insulin measurements of 0 indicate that your patient is dead. Testing the deceased for Diabetes is not likely to be of much interest, ergo, missing data.

We are also missing a Glucose measurement for one patient who has other measurements, but I will ignore this and filter only on 2-hr serum insulin, as it is the predominate indicator of missing data. I am making the assumption that missing data cannot be reasonably accomodated by replacement with the mean or some other value, thus will merely be eliminating those instances from consideration entirely.

Thus, the next step in improving upon the model is to filter data. As mentioned, I believe triceps skin fold measurements lack independence. First I will test this hypothesis, which will require an additional library to conduct a chi-square test. The null hypothesis is that there is no relationship, and I will test is alpha of 0.05.

In [6]:
```
from scipy import stats

df = pd.read_csv('./Diabetes Dataset/diabetes.csv', skiprows=0)
bin_diabetes = []
for i in df['Diabetes']:
    if i == 'tested_positive':
        bin_diabetes.append(1)
    else:
        bin_diabetes.append(0)
del df['Diabetes']
df['Diabetes_num'] = bin_diabetes

tricep,bmi = [],[]
for i in df['triceps skin fold thickness']:
    if i>0:
        tricep.append(i)
    else: tricep.append(0.0000000001)
for i in df['BMI']:
    if i>0:
        bmi.append(i)
    else: bmi.append(0.0000000001)

stats.pearsonr(tricep,bmi)
stats.chi2_contingency([tricep,bmi])
```

Out[6]:
```
(5979.821651568176,
 0.0,
 767,
 array([[26.81947237, 21.73706507,  9.1092377 , ..., 19.23495686,
         11.76772767, 24.00460063],
        [41.78052763, 33.86293493, 14.1907623 , ..., 29.96504314,
         18.33227233, 37.39539937]]))
```

Given that the P-value is well below 0.05 we can conclude that these two features related. Now we can trim the dataset by a feature and further filter the values of 2-hr serum == 0, though the later is merely data cleaning and not supported by any statistical hypothesis.

Note that this cell cannot be repeated: you must reload the prior cell each time you wish to run the cell below.

In [7]:
```
df = df.loc[(df['2hr-serum-insulin']>0)]
del df['triceps skin fold thickness']
df.head()
```

Out[7]:

	timesPreg	Glucose	blood pressure	2hr-serum-insulin	BMI	Diabetes pedigree fnc	age	Diabetes_num
3	1	89	66	94	28.1	0.167	21	0
4	0	137	40	168	43.1	2.288	33	1
6	3	78	50	88	31.0	0.248	26	1
8	2	197	70	543	30.5	0.158	53	1
13	1	189	60	846	30.1	0.398	59	1

Now we will reconstruct the prior test of Naïve Bayes and see if the results improve.

In [8]:
```
cols = list(df.columns)
cols.remove('Diabetes_num')
cols_z = []
for i in cols:
    col_zscore = i + '_z'
    cols_z.append(col_zscore)
    df[col_zscore] = (df[i] - df[i].mean())/df[i].std(ddof=1)
    del df[i]
rand_vals = []
for i in range(len(df)):
    rand_vals.append(random.random())
df['random']=rand_vals
overall =0
for i in range(k):
    rand_min,rand_max = i/k,(i+1)/k
    df_test  = df.loc[(df['random']>=rand_min) & (df['random']< rand_max
)]
    df_train = df.loc[(df['random']< rand_min) | (df['random']>=rand_max
)]
    model = sklearn.naive_bayes.GaussianNB()
    features = df_train[cols_z]
    labels = df_train['Diabetes_num']
    model.fit(features,labels)
    correct = 0
    for index,row in df_test.iterrows():
        row_data = []
        for i in row:
            row_data.append(i)
        actual = row_data.pop(0)
        del row_data[-1]
        classified = model.predict([row_data])
        if actual - classified[0]  ==0:
            correct+=1
    overall+=correct/len(df_test['Diabetes_num'])
overall=overall/k
print('Overall Accuracy with K='+str(k)+' fold validation:',overall)
```

```
Overall Accuracy with K=10 fold validation: 0.760737879662054
```

This is relatively close, and given that a 95% confidence interval gives approximately +/- 0.8% and +/- 1.5% for these sample sizes and means, the means are not necessarily different.

I would conclude, then, that a Naïve Bayes can classify diabetes diagnosis from given features about 75% of the time, and I am 95% confident that the true mean accuracy is within 1.5% of that number, unless further statistical analysis or transformation is conducted.