ILPD example from OpenML

1. Age Age of the patient
2. Gender Gender of the patient
3. TB Total Bilirubin
4. DB Direct Bilirubin
5. Alkphos Alkaline Phosphotase
6. Sgpt Alamine Aminotransferase
7. Sgot Aspartate Aminotransferase
8. TP Total Protiens
9. ALB Albumin
10. A/G Ratio Albumin and Globulin Ratio
11. Selector field used to split the data into two sets (labeled by the experts) NOTE: 1 = patient, 2 = control

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sn
from sklearn import neighbors
from sklearn.datasets import fetch_openml
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

ilpd = fetch_openml(name='ilpd', version=1)
ilpd.details
```

Out[1]:


```
{'id': '1480',
 'name': 'ilpd',
 'version': '1',
 'format': 'ARFF',
 'upload_date': '2015-05-22T22:40:56',
 'licence': 'Public',
 'url': 'https://www.openml.org/data/v1/download/1590565/ilpd.arff',
 'file_id': '1590565',
 'default_target_attribute': 'Class',
 'tag': ['health',
  'OpenML-CC18',
  'OpenML100',
  'study_123',
  'study_135',
  'study_14',
  'study_144',
  'study_34',
  'study_50',
  'study_52',
  'study_7',
  'study_98',
  'study_99',
  'uci'],
 'visibility': 'public',
 'status': 'active',
 'processing_date': '2018-10-03 21:42:36',
 'md5_checksum': '155295549a459387fb71312caf1b8360'}
```


In [2]:

```
X = pd.DataFrame(ilpd.data, columns=ilpd.feature_names)
Y = pd.Series(ilpd.target)
```

In [3]:

```
X
```

Out[3]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
0	65.0	0.0	0.7	0.1	187.0	16.0	18.0	6.8	3.3	0.90
1	62.0	1.0	10.9	5.5	699.0	64.0	100.0	7.5	3.2	0.74
2	62.0	1.0	7.3	4.1	490.0	60.0	68.0	7.0	3.3	0.89
3	58.0	1.0	1.0	0.4	182.0	14.0	20.0	6.8	3.4	1.00
4	72.0	1.0	3.9	2.0	195.0	27.0	59.0	7.3	2.4	0.40
...	...	...	...	...	...	...	...	...	...	...
578	60.0	1.0	0.5	0.1	500.0	20.0	34.0	5.9	1.6	0.37
579	40.0	1.0	0.6	0.1	98.0	35.0	31.0	6.0	3.2	1.10
580	52.0	1.0	0.8	0.2	245.0	48.0	49.0	6.4	3.2	1.00
581	31.0	1.0	1.3	0.5	184.0	29.0	32.0	6.8	3.4	1.00
582	38.0	1.0	1.0	0.3	216.0	21.0	24.0	7.3	4.4	1.50

583 rows × 10 columns

In [4]:

```
X.describe()
```

Out[4]:

	V1	V2	V3	V4	V5	V6	V
count	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.00000
mean	44.746141	0.756432	3.298799	1.486106	290.576329	80.713551	109.91080
std	16.189833	0.429603	6.209522	2.808498	242.937989	182.620356	288.91852
min	4.000000	0.000000	0.400000	0.100000	63.000000	10.000000	10.00000
25%	33.000000	1.000000	0.800000	0.200000	175.500000	23.000000	25.00000
50%	45.000000	1.000000	1.000000	0.300000	208.000000	35.000000	42.00000
75%	58.000000	1.000000	2.600000	1.300000	298.000000	60.500000	87.00000
max	90.000000	1.000000	75.000000	19.700000	2110.000000	2000.000000	4929.00000

In [5]:

```
Y
```

Out[5]:

```
0      1
1      1
2      1
3      1
4      1
      ..
578    2
579    1
580    1
581    1
582    2
Length: 583, dtype: object
```

In [6]:

```
Y.describe()
```

Out[6]:

```
count     583
unique      2
top         1
freq      416
dtype: object
```

In [7]:

```
Y.value_counts()
```

Out[7]:

```
1    416
2    167
dtype: int64
```

In [8]:

```
X.dtypes
```

Out[8]:

```
V1     float64
V2     float64
V3     float64
V4     float64
V5     float64
V6     float64
V7     float64
V8     float64
V9     float64
V10    float64
dtype: object
```

In [9]:

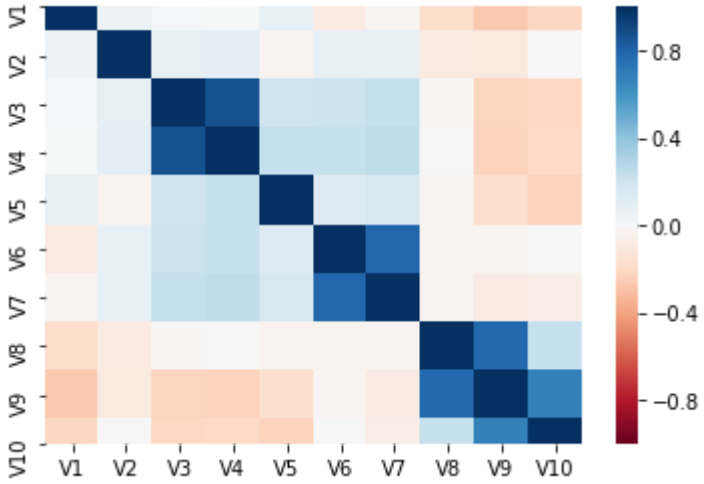
```
## Check for correlations
```

In [10]:

```
corrmatrix = X.corr()
sn.heatmap(corrmatrix, vmin=-1.0, vmax=1.0, cmap="RdBu")
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bbac660a88>
```



Drop correlated variables

The correlation plot shows heavy correlation between pairs of variables:

- V3 and V4
- V6 and V7
- V8 and V9

Uncomment X.drop to remove the latter of each pair so that the effects of those variables are not "double-counted".

In [11]:

```
# X.drop(["V4","V7","V9"], axis=1, inplace=True)
X
```

Out[11]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
0	65.0	0.0	0.7	0.1	187.0	16.0	18.0	6.8	3.3	0.90
1	62.0	1.0	10.9	5.5	699.0	64.0	100.0	7.5	3.2	0.74
2	62.0	1.0	7.3	4.1	490.0	60.0	68.0	7.0	3.3	0.89
3	58.0	1.0	1.0	0.4	182.0	14.0	20.0	6.8	3.4	1.00
4	72.0	1.0	3.9	2.0	195.0	27.0	59.0	7.3	2.4	0.40
...	...	...	...	...	...	...	...	...	...	...
578	60.0	1.0	0.5	0.1	500.0	20.0	34.0	5.9	1.6	0.37
579	40.0	1.0	0.6	0.1	98.0	35.0	31.0	6.0	3.2	1.10
580	52.0	1.0	0.8	0.2	245.0	48.0	49.0	6.4	3.2	1.00
581	31.0	1.0	1.3	0.5	184.0	29.0	32.0	6.8	3.4	1.00
582	38.0	1.0	1.0	0.3	216.0	21.0	24.0	7.3	4.4	1.50

583 rows × 10 columns

Standardize explanatory variables

Standardization makes the variables contribute with equal weights.

In [12]:

```
sc = StandardScaler()
sc.fit(X)
X = sc.transform(X)
```

Apply kNN

In [13]:

```
num_neighbors = 5

cf = neighbors.KNeighborsClassifier(n_neighbors=num_neighbors)
cf.fit(X,Y)
```

Out[13]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [14]:

```
cf.score(X,Y)
```

Out[14]:

```
0.8027444253859348
```

In [15]:

```
print(X)
```

```
[[ 1.25209764 -1.76228085 -0.41887783 ...  0.29211961  0.19896867
  -0.14789799]
 [ 1.06663704  0.56744644  1.22517135 ...  0.93756634  0.07315659
  -0.65069686]
 [ 1.06663704  0.56744644  0.6449187  ...  0.47653296  0.19896867
  -0.17932292]
 ...
 [ 0.44843504  0.56744644 -0.4027597  ... -0.0767071   0.07315659
   0.16635131]
 [-0.84978917  0.56744644 -0.32216906 ...  0.29211961  0.32478075
   0.16635131]
 [-0.41704777  0.56744644 -0.37052344 ...  0.75315299  1.58290153
   1.73759778]]
```

Validate

Add split validation

In [16]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33)
cf = neighbors.KNeighborsClassifier(n_neighbors=num_neighbors)
cf.fit(X_train, Y_train)
Y_pred = cf.predict(X_test)
```

Output predictions

In [17]:

```
Y_pred
```

Out[17]:

```
array(['2', '2', '2', '2', '2', '2', '1', '1', '2', '1', '1', '1', '1',
       '1', '1', '1', '1', '1', '1', '2', '1', '2', '2', '1', '1', '1',
       '1', '1', '2', '1', '1', '1', '1', '1', '1', '1', '1', '2', '1',
       '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '2', '1', '1',
       '1', '1', '1', '1', '2', '1', '1', '1', '1', '2', '1', '2', '1',
       '1', '1', '1', '1', '1', '2', '1', '2', '2', '1', '1', '2', '1',
       '1', '1', '1', '1', '1', '2', '2', '1', '1', '1', '1', '2', '1',
       '1', '1', '1', '1', '2', '2', '1', '1', '1', '1', '1', '1', '2',
       '1', '1', '1', '1', '2', '2', '1', '1', '1', '1', '1', '2', '2',
       '1', '1', '2', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
       '1', '1', '1', '1', '1', '1', '1', '2', '1', '1', '1', '2', '1',
       '1', '1', '2', '1', '1', '1', '1', '2', '1', '2', '1', '2', '1',
       '1', '2', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
       '2', '2', '1', '1', '2', '2', '1', '1', '2', '1', '2', '2', '1',
       '1', '1', '1', '2', '2', '1', '1', '1', '1', '1', '2'],
      dtype=object)
```

Output performance metrics

In [18]:

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(Y_test, Y_pred))
print(classification_report(Y_test, Y_pred))
print(cf.score(X_test,Y_test))
```

```
[[112  30]
 [ 33  18]]
              precision    recall  f1-score   support

           1       0.77      0.79      0.78       142
           2       0.38      0.35      0.36        51

    accuracy                           0.67       193
   macro avg       0.57      0.57      0.57       193
weighted avg       0.67      0.67      0.67       193

0.6735751295336787
```

Observations

In the data set, the proportion of affected individuals is 416/583 = 71,4% The trivial classifier (classify averyone as affected) can reach this accuracy.

The accuracy falls below this, so the method as current is not feasible.

However, the precision for the affected is higher than 71,4% A positive test result might still be a minor indication of a presence of the cancer.

In [ ]:

```
```