dataset

November 19, 2021

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.impute import KNNImputer
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, precision_score, recall_score
from sklearn.preprocessing import StandardScaler
# Import the models
from sklearn.ensemble import RandomForestClassifier
import lightgbm as lgb
from sklearn.svm import SVC
```

1 Explore the dataset

[ ]:
```
data = pd.read_csv('./heart.csv')
data
```

[ ]:
```
     Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  \
0     40   M           ATA        140          289          0     Normal
1     49   F           NAP        160          180          0     Normal
2     37   M           ATA        130          283          0         ST
3     48   F           ASY        138          214          0     Normal
4     54   M           NAP        150          195          0     Normal
..   ...  ..           ...        ...          ...        ...        ...
913   45   M            TA        110          264          0     Normal
914   68   M           ASY        144          193          1     Normal
915   57   M           ASY        130          131          0     Normal
916   57   F           ATA        130          236          0        LVH
917   38   M           NAP        138          175          0     Normal

     MaxHR ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0      172              N      0.0       Up             0
1      156              N      1.0     Flat             1
2       98              N      0.0       Up             0
3      108              Y      1.5     Flat             1
4      122              N      0.0       Up             0
..     ...            ...      ...      ...           ...
```

```
913    132              N      1.2     Flat             1
914    141              N      3.4     Flat             1
915    115              Y      1.2     Flat             1
916    174              N      0.0     Flat             1
917    173              N      0.0       Up             0

[918 rows x 12 columns]
```

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

[ ]:
```
              Age   RestingBP  Cholesterol   FastingBS       MaxHR  \
count  918.000000  918.000000   918.000000  918.000000  918.000000
mean    53.510893  132.396514   198.799564    0.233115  136.809368
std      9.432617   18.514154   109.384145    0.423046   25.460334
min     28.000000    0.000000     0.000000    0.000000   60.000000
25%     47.000000  120.000000   173.250000    0.000000  120.000000
50%     54.000000  130.000000   223.000000    0.000000  138.000000
75%     60.000000  140.000000   267.000000    0.000000  156.000000
max     77.000000  200.000000   603.000000    1.000000  202.000000

          Oldpeak  HeartDisease
count  918.000000    918.000000
mean     0.887364      0.553377
std      1.066570      0.497414
min     -2.600000      0.000000
25%      0.000000      0.000000
50%      0.600000      1.000000
75%      1.500000      1.000000
max      6.200000      1.000000
```

[ ]:
```
for cat in data:
    if len(data[cat].unique()) < 20:
        print(cat, data[cat].unique())
```

```
Sex ['M' 'F']
ChestPainType ['ATA' 'NAP' 'ASY' 'TA']
FastingBS [0 1]
RestingECG ['Normal' 'ST' 'LVH']
ExerciseAngina ['N' 'Y']
ST_Slope ['Up' 'Flat' 'Down']
HeartDisease [0 1]
```

2 Apply one hot encoding on catagorical data

[ ]:
```
data = pd.get_dummies(data, drop_first=False)
data.drop(['ST_Slope_Flat', 'Sex_F', 'ExerciseAngina_N', 'RestingECG_Normal'],␣
 ↪axis=1)
```

[ ]:
```
      Age  RestingBP  Cholesterol  FastingBS  MaxHR  Oldpeak  HeartDisease  \
0      40        140          289          0    172      0.0             0
1      49        160          180          0    156      1.0             1
2      37        130          283          0     98      0.0             0
3      48        138          214          0    108      1.5             1
4      54        150          195          0    122      0.0             0
..    …          …            …          …    …        …               …
913    45        110          264          0    132      1.2             1
914    68        144          193          1    141      3.4             1
915    57        130          131          0    115      1.2             1
916    57        130          236          0    174      0.0             1
917    38        138          175          0    173      0.0             0

      Sex_M  ChestPainType_ASY  ChestPainType_ATA  ChestPainType_NAP  \
0         1                  0                  1                  0
1         0                  0                  0                  1
2         1                  0                  1                  0
3         0                  1                  0                  0
4         1                  0                  0                  1
..      …                  …                  …                  …
913       1                  0                  0                  0
914       1                  1                  0                  0
915       1                  1                  0                  0
916       0                  0                  1                  0
917       1                  0                  0                  1

      ChestPainType_TA  RestingECG_LVH  RestingECG_ST  ExerciseAngina_Y  \
0                    0               0              0                 0
1                    0               0              0                 0
2                    0               0              1                 0
3                    0               0              0                 1
4                    0               0              0                 0
..                 …               …              …                 …
913                  1               0              0                 0
914                  0               0              0                 0
915                  0               0              0                 1
916                  0               1              0                 0
917                  0               0              0                 0

      ST_Slope_Down  ST_Slope_Up
0                 0            1
```

```
1                0              0
2                0              1
3                0              0
4                0              1
..             …              …
913              0              0
914              0              0
915              0              0
916              0              0
917              0              1

[918 rows x 17 columns]
```

3 Use K-nearest neighbour for missing cholesterol values

[ ]:
```
data.Cholesterol = data.Cholesterol.replace({0:np.nan})
imputer = KNNImputer(n_neighbors=5)
data = pd.DataFrame(imputer.fit_transform(data),columns = data.columns)
```

[ ]:
```
def evaluate(pred, y_test):
    print('acc:', (pred == y_test).sum()/len(pred))
    print('precision:', precision_score(pred, y_test))
    print('recall:', recall_score(pred, y_test))
    print('f1:', f1_score(pred, y_test))
```

4 Normalize the data

[ ]:
```
X = data.drop('HeartDisease', axis = 1).values
y = data['HeartDisease'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.15,␣
 ↪random_state = 42, stratify=y)

scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

5 Random Forest

[ ]:
```
rf = RandomForestClassifier(n_estimators=500, random_state=42)
rf.fit(X_train, y_train)
```

[ ]:
```
RandomForestClassifier(n_estimators=500, random_state=42)
```

[ ]:
```
pred = rf.predict(X_test)
evaluate(pred, y_test)
```

```
acc: 0.8840579710144928
precision: 0.8947368421052632
recall: 0.8947368421052632
f1: 0.8947368421052632
```

6 Support vector machines

[ ]:
```
kernels=["linear","rbf","poly","sigmoid"]
#kernels=["rbf"]
for kernel in kernels:
    svm = SVC(kernel=kernel)
    svm.fit(X_train, y_train)
    pred=svm.predict(X_test)
    print('\n')
    print('kernel', kernel)
    evaluate(pred, y_test)
```

```
kernel linear
acc: 0.8695652173913043
precision: 0.8552631578947368
recall: 0.9027777777777778
f1: 0.8783783783783783


kernel rbf
acc: 0.8695652173913043
precision: 0.9078947368421053
recall: 0.8625
f1: 0.8846153846153847


kernel poly
acc: 0.8695652173913043
precision: 0.868421052631579
recall: 0.8918918918918919
f1: 0.88


kernel sigmoid
acc: 0.8695652173913043
precision: 0.868421052631579
recall: 0.8918918918918919
```

```
f1: 0.88
```

7 Tree based gradient boosting

[ ]:
```
clf = lgb.LGBMClassifier(n_estimators=40)
clf.fit(X_train, y_train)
```

[ ]:
```
LGBMClassifier(n_estimators=40)
```

[ ]:
```
pred = clf.predict(X_test)

evaluate(pred, y_test)
```

```
acc: 0.8913043478260869
precision: 0.881578947368421
recall: 0.9178082191780822
f1: 0.8993288590604027
```

8 Artificial Neural Network

[ ]:
```
import os
import torch
from torch import nn
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms
import pytorch_lightning as pl

class trainData(Dataset):
    def __init__(self, X_data, y_data):
        self.X_data = X_data
        self.y_data = y_data

    def __getitem__(self, index):
        return self.X_data[index], self.y_data[index]

    def __len__ (self):
        return len(self.X_data)

class testData(Dataset):
    def __init__(self, X_data):
        self.X_data = X_data

    def __getitem__(self, index):
        return self.X_data[index]

    def __len__ (self):
```

```
        return len(self.X_data)

class MLP(pl.LightningModule):

  def __init__(self, n_feats, n_out):
    super().__init__()
    self.layers = nn.Sequential(
      nn.Linear(n_feats, 64),
      nn.Dropout(0.1),
      nn.ReLU(),
      nn.Linear(64, 32),
      nn.Dropout(0.1),
      nn.ReLU(),

      nn.Linear(32, n_out)
    )
    self.criterion = nn.BCEWithLogitsLoss()

  def forward(self, x):
    return self.layers(x)

  def training_step(self, batch, batch_idx):
    x, y = batch
    x = x.view(x.size(0), -1)
    y_hat = self.layers(x)
    y = nn.functional.one_hot(y.to(torch.int64), 2).to(torch.float32)
    loss = self.criterion(y_hat.to(torch.float32), y)
    self.log('train_loss', loss)
    return loss

  def configure_optimizers(self):
    optimizer = torch.optim.Adam(self.parameters(), lr=1e-4, weight_decay=0.
↪0001)
    return optimizer
```

[ ]:

```
  dataset = trainData(X_train.astype(np.float32), y_train.astype(np.float32))
  pl.seed_everything(42)
  mlp = MLP(20 ,2)
  trainer = pl.Trainer(auto_scale_batch_size='power', gpus=0,␣
↪deterministic=True, max_epochs=3)
  trainer.fit(mlp, DataLoader(dataset))
```

```
Global seed set to 42
GPU available: True, used: False
TPU available: False, using: 0 TPU cores
```

```
IPU available: False, using: 0 IPUs
/home/nueron/miniconda3/lib/python3.9/site-
packages/pytorch_lightning/trainer/trainer.py:1566: UserWarning: GPU available
but not used. Set the gpus flag in your trainer `Trainer(gpus=1)` or script
`--gpus=1`.
  rank_zero_warn(

  | Name      | Type              | Params
-------------------------------------------------
0 | layers    | Sequential        | 3.5 K
1 | criterion | BCEWithLogitsLoss | 0
-------------------------------------------------
3.5 K     Trainable params
0         Non-trainable params
3.5 K     Total params
0.014     Total estimated model params size (MB)
/home/nueron/miniconda3/lib/python3.9/site-
packages/pytorch_lightning/trainer/data_loading.py:110: UserWarning: The
dataloader, train_dataloader, does not have many workers which may be a
bottleneck. Consider increasing the value of the `num_workers` argument` (try 32
which is the number of cpus on this machine) in the `DataLoader` init to improve
performance.
  rank_zero_warn(

Training: 0it [00:00, ?it/s]
```

[ ]:
```
test_tensor = torch.Tensor(X_test)
pred = mlp(test_tensor)

pred = torch.argmax(pred, axis=1).numpy()
evaluate(pred, y_test)
```

```
acc: 0.9057971014492754
precision: 0.881578947368421
recall: 0.9436619718309859
f1: 0.9115646258503401
```