gb_v_ls

February 5, 2024

[ ]:
```python
# Import libraries
import matplotlib.pyplot as plt
import sklearn.model_selection as sms
import sklearn.linear_model as slm
import sklearn.preprocessing as skp
import sklearn.metrics as sme
import sklearn.feature_selection as skf
import sklearn.ensemble as ske
import sklearn.utils as sku
import sklearn.decomposition as skd
import sklearn.neural_network as skn
from sklearnex import patch_sklearn, config_context
from sklearn.cluster import DBSCAN
import numpy as np
import scipy.stats as stats
from IPython.display import HTML
import util
from scipy.spatial import cKDTree
import loss_landscapes
import loss_landscapes.metrics
import copy
from torchviz import make_dot
from sklearn.ensemble import GradientBoostingRegressor
patch_sklearn()
```

```
Intel(R) Extension for Scikit-learn* enabled (https://github.com/intel/scikit-learn-intelex)
```

For samples $i \in N$ and features $j \in p$ with targets y_i, the optimal features β_{ij} can be determined by minimizing a loss function $l(y, f(X))$. The matrix X is formed by standardizing the feature vectors such that $\frac{1}{N}\sum_{i=1}^{N} x_{ij} = 0$ and $\frac{1}{N}\sum_{i=1}^{N} x_{ij}^2 = 1$. Then, solving for the optimal weights (least absolute shrinkage operator, LASSO) amounts to minimizing the negative likelihood of observing (y_i, X_i), or

$$\hat{\beta} = \text{argmin}_{\beta} \sum_{i=1}^{N} l(y_i, f(X_i)) + \lambda \sum_{j=1}^{p} |\beta_j|$$

Gradient boosting models the prediction as a weighted sum of base learners $h_i(x_i) such that

$f(X) = \beta_0 + h_1(x_1) + ... + h_p(x_p)$. The optimal learner combination is

$$\hat{f}^m(X) = \hat{f}^{m-1}(X) + \nu \cdot \hat{h}^m_{j^*}(x_{j^*}) \quad \text{s.t.} \quad j^* = \text{argmin}_{1 \le j \le p} \sum_{i=1}^{N} \left(\left(-\frac{\partial l(y_i, f(X_i))}{\partial f} \right) \Big|_{f=f^{m-1}(X_i)} - \hat{h}^m_j(x_{ij}) \right)^2$$

Below, feature vectors x and labels y are loaded and $X \sim \mathcal{N}(0,1)$ is generated

[ ]:
```
# Get case IDs
case_list = open('/home/ali/RadDBS-QSM/data/docs/cases_90','r')
lines = case_list.read()
lists = np.loadtxt(case_list.name,comments="#",␣
 ↪delimiter=",",unpack=False,dtype=str)
case_id = []
for lines in lists:
    case_id.append(lines[-9:-7])

# Load scores
file_dir = '/home/ali/RadDBS-QSM/data/docs/QSM anonymus- 6.22.2023-1528.csv'
motor_df = util.filter_scores(file_dir,'pre-dbs updrs','stim','CORNELL ID')
# Find cases with all required scores
subs,pre_imp,post_imp,pre_updrs_off = util.get_full_cases(motor_df,
                                                          'CORNELL ID',
                                                          'OFF (pre-dbs updrs)',
                                                          'ON (pre-dbs updrs)',
                                                          'OFF meds ON stim␣
 ↪6mo')
# Load extracted features
npy_dir = '/home/ali/RadDBS-QSM/data/npy/'
phi_dir = '/home/ali/RadDBS-QSM/data/phi/phi/'
roi_path = '/data/Ali/atlas/mcgill_pd_atlas/PD25-subcortical-labels.csv'
n_rois = 6
all_rois = False
Phi_all, X_all, R_all, K_all, ID_all = util.load_featstruct(phi_dir,npy_dir+'X/
 ↪',npy_dir+'R/',npy_dir+'K/',n_rois,1595,all_rois)
ids = np.asarray(ID_all).astype(int)
# Find overlap between scored subjects and feature extraction cases
c_cases = np.intersect1d(np.asarray(case_id).astype(int),np.asarray(subs).
 ↪astype(int))
# Complete case indices with respect to feature matrix
c_cases_idx = np.in1d(ids,c_cases)

X_all_c, K, R, subsc, pre_imp, pre_updrs_off, per_change = util.
 ↪re_index(X_all,K_all,R_all,c_cases_idx,subs,ids,all_rois,pre_imp,pre_updrs_off,post_imp)
```

```
Allocated arrays
Created feature matrix
Created ROI matrix
Created feature label matrix
```

```
['Left red nucleus' 'Left substantia nigra' 'Left subthalamic nucleus'
 'Right Substantia nigra' 'Right red nucleus' 'Right subthalamic nucleus']
```

Then, the low-dimension condition

$$p < N$$

Of the positive cone condition is imposed to ensure gradient boosting and LASSO estimators converge

[ ]:
```
p = X_all_c.shape[0]-2
```

Let S be a diagonal $p \times p$ matrix with elements $s_{ij} = s_{ii} \in \{-1, 1\}$ and X remains the feature matrix for the dataset, $N \times p$. The the positive cone condition is met if

$$\left(S'X'XS\right)^{-1}\mathbb{I}_{p\times 1}$$

For all subsets of X and possible combinations of S. A more tractable form is the diagonal dominance condition

$$|M_{jj}| \geq \sum_{i\neq j} |M_{ij}|$$

Here, M is the inverse covariance matrix of X, $M = (X'X)^{-1}$

[ ]:
```
# Training parameters
scoring = 'r2'
results_bls = np.zeros_like(per_change)
results_ls = np.zeros_like(per_change)
# Train
for j in np.arange(len(subsc)):
    test_id = subsc[j]
    test_index = subsc == test_id
    train_index = subsc != test_id
    X_train = X_all_c[train_index,:,:]
    X_test = X_all_c[test_index,:,:]
    y_train = per_change[train_index]
    y_test = per_change[test_index]
    # Cross validation
    cvn = 6
    X0_ss0,scaler_ss,X_test_ss0 = util.model_scale(skp.StandardScaler(),␣
 ↪X_train,train_index,X_test,test_index,pre_updrs_off,False)
    with np.errstate(divide='ignore', invalid='ignore'):
      # Feature selection
      sel = skf.SelectKBest(skf.r_regression,k=p)
      X0_ss = sel.fit_transform(X0_ss0,y_train)
      X_test_ss = sel.transform(X_test_ss0)
      y_n = cKDTree(X0_ss).query(X_test_ss, k=1)[1]

    # LASSO
    lasso = slm.Lasso(max_iter=1e6,alpha=1e-1)
    est_ls = lasso.fit(X0_ss,y_train)
```

```
    results_ls[j] = est_ls.predict(X_test_ss)

    # Gradient boosting
    gsc = sms.GridSearchCV(
            estimator=GradientBoostingRegressor(validation_fraction=0),
            param_grid={"learning_rate": [1e-2],
                        "max_depth": [1],
                        "n_estimators": [p]},
                        cv=cvn, scoring='neg_mean_squared_error', n_jobs=-1)
    est_gr = gsc.fit(X0_ss, y_train)
    results_bls[j] = est_gr.predict(X_test_ss)

    # Training status
    print('Lasso predicts',str(np.round(results_ls[j],2)),
          'and gradient boost predicts',str(np.round(results_bls[j],2)))#,'for␣
↪case',str(int(subsc[j])),'with',str(np.round(per_change[j],2)))
```

```
Lasso predicts 0.63 and gradient boost predicts 0.64
Lasso predicts 0.62 and gradient boost predicts 0.6
Lasso predicts 0.63 and gradient boost predicts 0.62
Lasso predicts 0.63 and gradient boost predicts 0.63
Lasso predicts 0.63 and gradient boost predicts 0.67
Lasso predicts 0.64 and gradient boost predicts 0.65
Lasso predicts 0.63 and gradient boost predicts 0.59
Lasso predicts 0.63 and gradient boost predicts 0.57
Lasso predicts 0.63 and gradient boost predicts 0.61
Lasso predicts 0.63 and gradient boost predicts 0.64
Lasso predicts 0.64 and gradient boost predicts 0.65
Lasso predicts 0.63 and gradient boost predicts 0.61
Lasso predicts 0.64 and gradient boost predicts 0.64
Lasso predicts 0.64 and gradient boost predicts 0.67
Lasso predicts 0.63 and gradient boost predicts 0.62
Lasso predicts 0.63 and gradient boost predicts 0.62
Lasso predicts 0.63 and gradient boost predicts 0.64
Lasso predicts 0.64 and gradient boost predicts 0.64
Lasso predicts 0.63 and gradient boost predicts 0.56
Lasso predicts 0.64 and gradient boost predicts 0.65
Lasso predicts 0.63 and gradient boost predicts 0.59
Lasso predicts 0.64 and gradient boost predicts 0.63
Lasso predicts 0.63 and gradient boost predicts 0.61
Lasso predicts 0.62 and gradient boost predicts 0.58
Lasso predicts 0.62 and gradient boost predicts 0.59
Lasso predicts 0.63 and gradient boost predicts 0.62
Lasso predicts 0.61 and gradient boost predicts 0.59
Lasso predicts 0.6 and gradient boost predicts 0.6
Lasso predicts 0.63 and gradient boost predicts 0.67
Lasso predicts 0.63 and gradient boost predicts 0.62
```

```
Lasso predicts 0.65 and gradient boost predicts 0.64
Lasso predicts 0.63 and gradient boost predicts 0.66
Lasso predicts 0.63 and gradient boost predicts 0.67
Lasso predicts 0.62 and gradient boost predicts 0.64
Lasso predicts 0.63 and gradient boost predicts 0.67
Lasso predicts 0.63 and gradient boost predicts 0.63
Lasso predicts 0.62 and gradient boost predicts 0.64
Lasso predicts 0.5 and gradient boost predicts 0.6
Lasso predicts 0.63 and gradient boost predicts 0.63
Lasso predicts 0.63 and gradient boost predicts 0.67
Lasso predicts 0.63 and gradient boost predicts 0.55
Lasso predicts 0.63 and gradient boost predicts 0.62
Lasso predicts 0.64 and gradient boost predicts 0.68
Lasso predicts 0.62 and gradient boost predicts 0.65
Lasso predicts 0.63 and gradient boost predicts 0.64
Lasso predicts 0.63 and gradient boost predicts 0.56
Lasso predicts 0.62 and gradient boost predicts 0.61
```

The mean squared error between the LASSO estimator $f(X, \hat{\beta})$ and the gradient boosting regressor $\hat{f}^m(X)$ is $\mathcal{O} \sim 10^{-3}$

[ ]:
```
np.mean((results_ls-results_bls)**2)
```

[ ]:
```
0.0010680490300377554
```