Introduction

The goal of this notebook is to provide some technical details (key steps) for our submitted manuscript:

> *Haolin Wang, et al. "Integrating Co-clustering and Interpretable Machine Learning for the Prediction of Intravenous Immunoglobulin Resistance in Kawasaki Disease", 2020.*

To improve the performance of clinical prediction models addressing the incompleteness of EHRs data, the proposed method performed co-clustering to address the incompleteness of clinical data, group Lasso for group-based feature selection, and Explainable Boosting Machine for group-specific prediction in a sequential manner.

Fig 1. The block-wise missing patterns characterized by co-clustering.

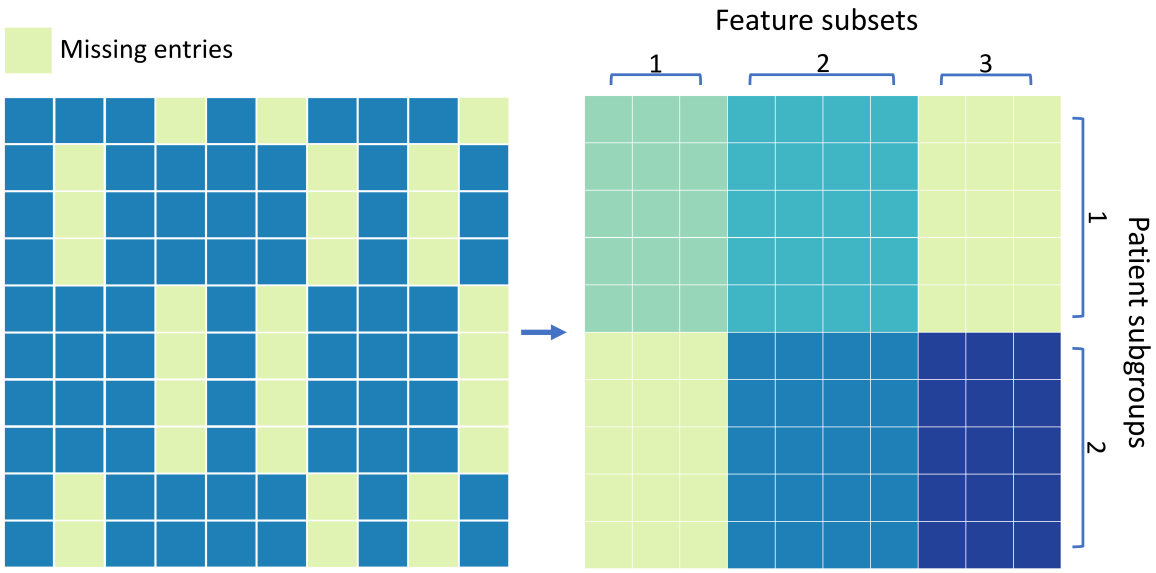
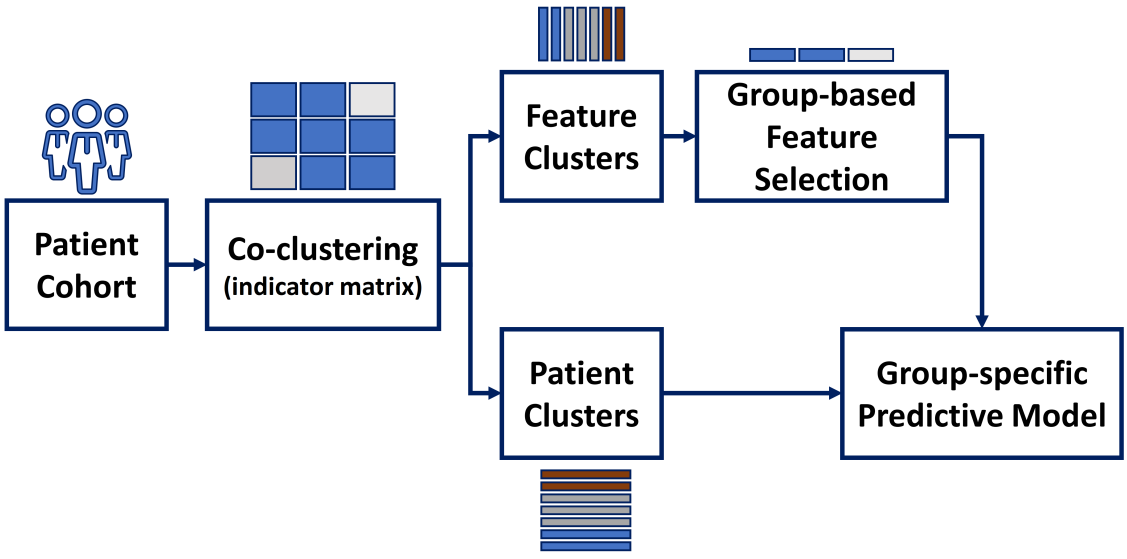


Fig 2. The proposed multiple classifier system with static classifier selection based on co-clustering.



Tools and Implementation

Packages

- Popular Python machine learning libraries such as scikit-learn and pandas
- imbalanced-learn: https://github.com/scikit-learn-contrib/imbalanced-learn
- InterpretML: https://github.com/interpretml/interpret
- Coclust: https://pypi.org/project/coclust/
- Group Lasso: https://github.com/AnchorBlues/GroupLasso/blob/master/grouplasso/model.py

Preprocessing

```
import pandas as pd

dat = pd.read_csv('dataset.csv')
df = pd.get_dummies(dat, columns=['categories'])
```

Over-sampling for imbalanced dataset

```
from imblearn.over_sampling import SMOTE

balanced_feature_set, balanced_label = SMOTE(sampling_strategy=<>,
random_state=0).fit_resample(feature, label)
```

Missing data imputation to address data missing at random

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

imp = IterativeImputer(max_iter=5, random_state=0, tol=0.005)
feature_set = imp.fit_transform(feature_set)
```

Dataset splitting for cross-validation

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=5)
skf.get_n_splits(X, y)
```

Baseline methods

```
#Lasso
from sklearn.linear_model import Lasso
#Logistic
from sklearn.linear_model import LogisticRegression
#Ridge
from sklearn.linear_model import Ridge
#KNN
from sklearn import neighbors
model = neighbors.KNeighborsClassifier()
#Naive Bayes
from sklearn.naive_bayes import GaussianNB
#MLP
from sklearn.neural_network import MLPClassifier
#Random forest
```

```
from sklearn.ensemble import RandomForestClassifier
#lightGBM
import lightgbm
#XGBoost
import xgboost as xgb
#EBM
from interpret.glassbox import ExplainableBoostingClassifier

#Those models are trained and tested using the same splitted dataset for cross-validation. The usage of lightGBM is slightly different with the sklearn models.
train_data = lgb.Dataset(train_set, label=train_label)
param = {'metrics':'auc', 'objective': 'binary'}
model = lgb.train(param, train_data)
prob = model.predict(test_set)
```

Tuning the hyper-parameters

```
#alpha for Lasso
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV

params = {"alpha": numpy.logspace(-3, 1, 5)}
model_cv = GridSearchCV(Lasso(), params, cv=5)
model = model_cv.fit(train_set, train_label)
print("tuned hpyerparameters:", model.best_params_)
prob = model_cv.predict(test_set)
```

Co-clustering and Classification

Indicator matrix for incomplete dataset

```
# generate indicator matrix using the original dataset without missing data imputation
[rows, cols] = train_set_original.shape
train_ind = numpy.zeros((rows, cols))
for r in range(rows):
    for c in range(cols):
        if not numpy.isnan(train_set_original[r, c]):
            train_ind[r, c] = 1
train_rows_num = rows

...

# merge training set and test set splitted for cross-validation
full_ind = numpy.row_stack((train_ind, test_ind))
```

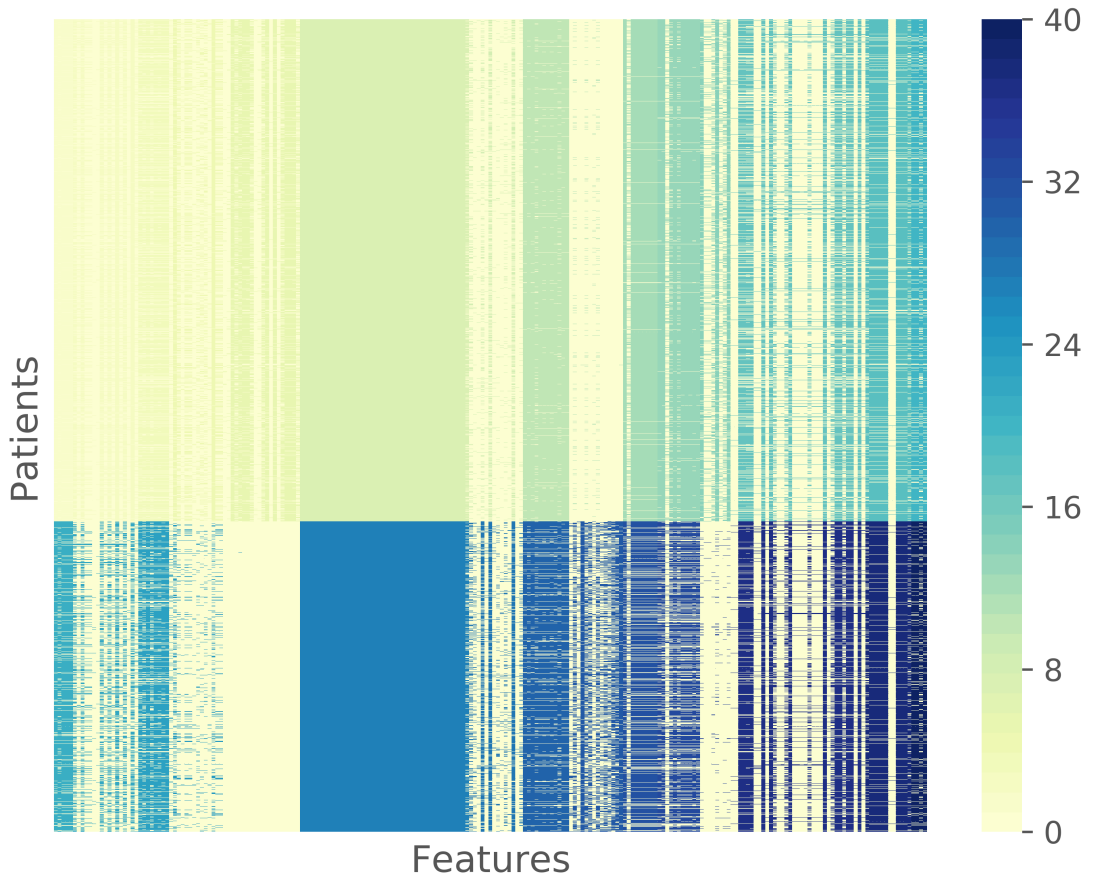
Co-clustering of the indicator matrix

```
from coclust.coclustering import CoclustInfo

for row_cluster in range(2, 10):
    for column_cluster in range(2,30):
        ...
        model =
CoclustInfo(n_row_clusters=row_cluster,n_col_clusters=column_cluster,
random_state=42)
        model.fit(full_ind)
        row_labels = model.row_labels_
        print(row_labels)
        col_labels = model.column_labels_
        print(col_labels)
        ...
```

Visualization

```
row_indices = numpy.argsort(model.row_labels_)
col_indices = numpy.argsort(model.column_labels_)
X_reorg = full_ind[row_indices, :]
X_reorg = X_reorg[:, col_indices]
cmap = sns.color_palette("YlGnBu", 41)
fig = sns.heatmap(X_reorg, cmap=cmap, xticklabels=False, yticklabels=False)
plt.savefig('co-cluster.tif', dpi=600, format='tif')
```



Re-organize samples for each row clusters to train multiple classifiers

```
for cluster_index in range(0, row_cluster):
    co_train_set = []
    co_train_label = []
    for k in range(0, train_rows_num):
        if row_labels[k] == cluster_index:
            co_train_label.append(train_label[k])
            co_train_set.append(train_set[k,:])
    co_train_set = numpy.array(co_train_set)
    co_train_label = numpy.array(co_train_label)
    print(co_train_set.shape)
    print(co_train_label.shape)
```

Feature selection with group feature structure

Group Lasso derives feature coefficients from certain groups to be small or exact zero.

```
# train group lasso
from grouplasso import GroupLassoClassifier

model =
GroupLassoClassifier(group_ids=numpy.array(col_labels),alpha=alpha_test,
eta=0.001, tol=0.001, max_iter=3000, random_state=42, verbose_interval=300)
model.fit(co_train_set, co_train_label)
print(model.coef_)

# feature selection
selected_cols = []
for k in range(0, len(col_labels)):
    if abs(model.coef_[k]) > 0:
        selected_cols.append(k)
```

Train group-specific prediction model

```
decision_model = ExplainableBoostingClassifier().fit(co_train_set[:,
selected_cols], co_train_label)
predict_test += list(decision_model.predict_proba(co_test_set[:,
selected_cols])[:,1])
predict_test_label += list(co_test_label)
```

Evaluation metrics

```
from sklearn.metrics import precision_recall_fscore_support

#for 5-fold cross-validation
for index in range(1, 6):
    best_f1 = 0
    keep_score = []
    test_label = np.load('xxx.npy')
    pred = np.load('xxx.npy')
    fpr, tpr, thresholds = metrics.roc_curve(test_label, prob)
    roc_auc = metrics.auc(fpr, tpr)
    print(roc_auc)

    for thres in pred:
        res_bin = []
```

```
        for p in prob:
            if p >= thres:
                res_bin.append(1)
            else:
                res_bin.append(0)

        score = precision_recall_fscore_support(test_label, res_bin,
average='binary')
        print(score)
        if score[2] > best_f1:
            keep_score = score
            best_f1 = score[2]
    best_score.append(keep_score)
    average_pr.append((keep_score[0] + keep_score[1])/2)

print(best_score)
best_score_array = np.array(best_score)
average_pr = np.array(average_pr)

print( str(np.around(best_score_array[:,0].mean(), decimals=3)) + '+' +
str(np.around(best_score_array[:,0].std(), decimals=3)))
print( str(np.around(best_score_array[:,1].mean(), decimals=3)) + '+' +
str(np.around(best_score_array[:,1].std(), decimals=3)))
print( str(np.around(best_score_array[:,2].mean(), decimals=3)) + '+' +
str(np.around(best_score_array[:,2].std(), decimals=3)))
print( str(np.around(average_pr.mean(), decimals=3)) + '+' +
str(np.around(average_pr.std(), decimals=3)))
```

Interpretability

```
model = ExplainableBoostingClassifier()
model.fit(train_set, train_label)

ebm_global = model.explain_global()
# show(ebm_global)
# export model parameters
pd.DataFrame(ebm_global._internal_obj['overall']).to_csv('xxx.csv',
index=False, header=False)

ebm_local = model.explain_local(train_set, train_label)
# export model parameters
pd.DataFrame(ebm_local._internal_obj['specific'][<sample_id>]
['scores']).to_csv('xxx.csv', index=False, header=False)
```