Few utility we have created:

Ensemble weighted voting rule utilities for model selections:

# -\*- coding: utf-8 -\*-

"""

Created on Mon Jun 27 13:45:27 2016

@author: sridhar.jeyaraman

"""

from sklearn.ensemble import AdaBoostClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import GradientBoostingClassifier

from sklearn import cross\_validation

from sklearn.ensemble import VotingClassifier

from sklearn.linear\_model import LogisticRegression

import pandas as pd

import numpy as np

cfl1 = AdaBoostClassifier(base\_estimator=None,n\_estimators=350, algorithm = 'SAMME.R')

cfl2 = RandomForestClassifier(n\_estimators=350, criterion = 'entropy',max\_features=None,n\_jobs=-1,warm\_start=True)

cfl3 = GradientBoostingClassifier(learning\_rate=0.20)

cfl4 = LogisticRegression()

elf1=VotingClassifier(estimators=[('ad',cfl1),('rf',cfl2),('GB',cfl3),('LR',cfl4)],voting='hard')

elf2=VotingClassifier(estimators=[('ad',cfl1),('rf',cfl2),('GB',cfl3),('LR',cfl4)],voting='soft')

elf2=VotingClassifier(estimators=[('ad',cfl1),('rf',cfl2),('GB',cfl3),('LR',cfl4)],voting='soft',weights=[0.33,0.34,0.33])

print('5-fold cross validation:\n')

for clf, label in zip([cfl1, cfl2, cfl3,elf1,elf2], ['Ada Boost', 'Random Forest', 'Gradient Boost','Logistic Regression', 'Ensemble hard', 'Ensemble soft', 'Ensemble weighted']):

scores = cross\_validation.cross\_val\_score(clf, x\_train, y\_train, cv=5, scoring='accuracy')

print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))

df = pd.DataFrame(columns=('w1', 'w2', 'w3','w4', 'mean', 'std'))

i = 0.0

for w1 in np.linspace(0.1,1.0,10):

for w2 in np.linspace(0.1,1.0,10):

for w3 in np.linspace(0.1,1.0,10):

for w4 in np.linspace(0.1,1.0,10):

if len(set((w1,w2,w3,w4))) == 1:

continue

eclf = VotingClassifier(estimators=[('ad',cfl1),('rf',cfl2),('GB',cfl3),('LR',cfl4)], weights=[w1,w2,w3,w4])

scores = cross\_validation.cross\_val\_score(

estimator=eclf,

X=x\_train,

y=y\_train,

cv=5,

scoring='accuracy',

n\_jobs=1)

df.loc[i] = [w1, w2, w3, scores.mean(), scores.std()]

i += 1

df.sort(columns=['mean', 'std'], ascending=False)

Utilities for Hyper Parameter tuning using grid and random search:

# -\*- coding: utf-8 -\*-

"""

Created on Thu Jun 16 13:09:30 2016

@author: sridhar.jeyaraman

"""

print(\_\_doc\_\_)

import numpy as np

from time import time

from operator import itemgetter

from scipy.stats import randint as sp\_randint

from sklearn.grid\_search import GridSearchCV, RandomizedSearchCV

from sklearn.ensemble import RandomForestClassifier

# get some data

X, y = x\_train, y\_train

# build a classifier

clf = RandomForestClassifier(n\_estimators=300)

# Utility function to report best scores

def report(grid\_scores, n\_top=3):

top\_scores = sorted(grid\_scores, key=itemgetter(1), reverse=True)[:n\_top]

for i, score in enumerate(top\_scores):

print("Model with rank: {0}".format(i + 1))

print("Mean validation score: {0:.3f} (std: {1:.3f})".format(

score.mean\_validation\_score,

np.std(score.cv\_validation\_scores)))

print("Parameters: {0}".format(score.parameters))

print("")

# specify parameters and distributions to sample from

param\_dist = {"max\_depth": [15, None],

"max\_features": sp\_randint(35, 50),

"min\_samples\_split": sp\_randint(1, 15),

"min\_samples\_leaf": sp\_randint(1, 15),

"bootstrap": [True, False],

"criterion": ["gini", "entropy"]}

# run randomized search

n\_iter\_search = 25

random\_search = RandomizedSearchCV(clf, param\_distributions=param\_dist,

n\_iter=n\_iter\_search)

start = time()

random\_search.fit(X, y)

print("RandomizedSearchCV took %.2f seconds for %d candidates"

" parameter settings." % ((time() - start), n\_iter\_search))

report(random\_search.grid\_scores\_)

# use a full grid over all parameters

param\_grid = {"max\_depth": [15, None],

"max\_features": [35, 55, 67],

"min\_samples\_split": [10, 15, 25],

"min\_samples\_leaf": [25, 45, 50],

"bootstrap": [True, False],

"criterion": ["gini", "entropy"]}

# run grid search

grid\_search = GridSearchCV(clf, param\_grid=param\_grid)

start = time()

grid\_search.fit(X, y)

print("GridSearchCV took %.2f seconds for %d candidate parameter settings."

% (time() - start, len(grid\_search.grid\_scores\_)))

report(grid\_search.grid\_scores\_)