! This class has been made inactive. No posts will be allowed until an instructor reactivates the class.

private note @64 3 views

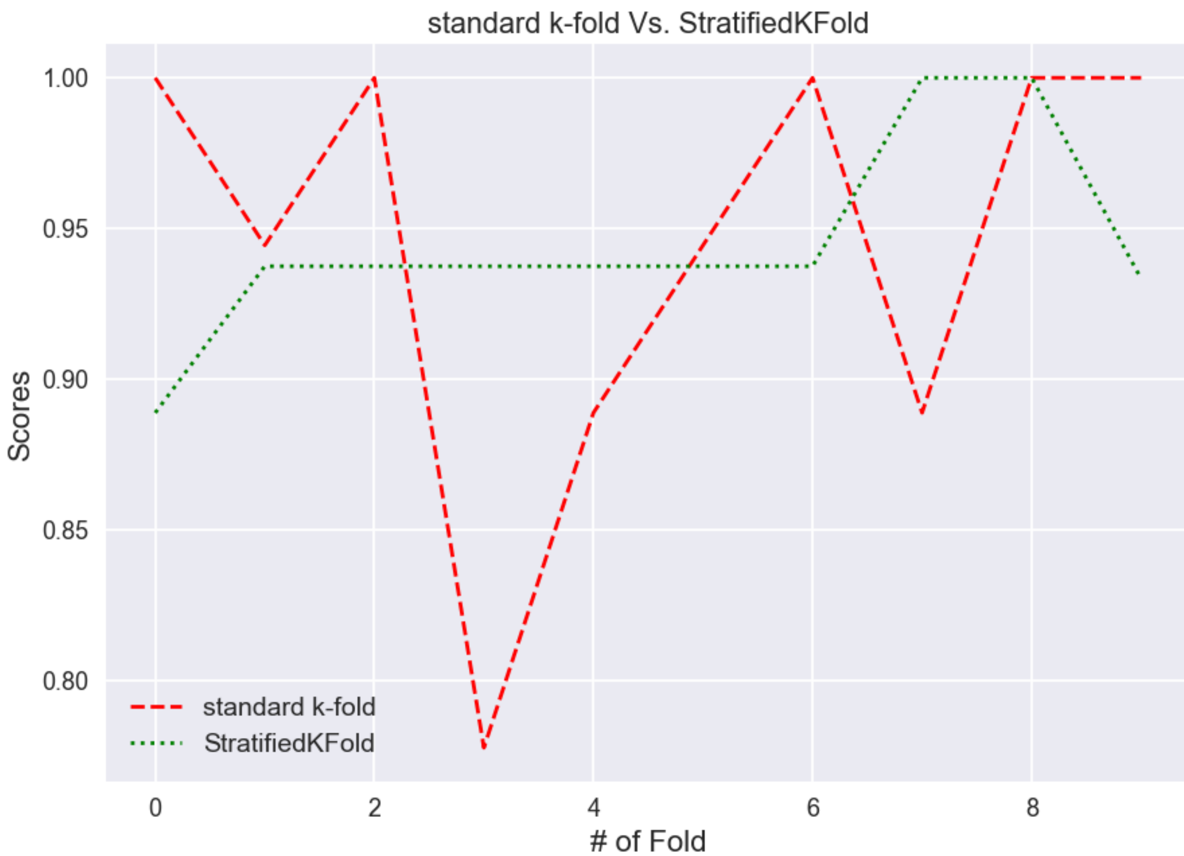
HW3_2017310936_Md_Shirajum_Munir

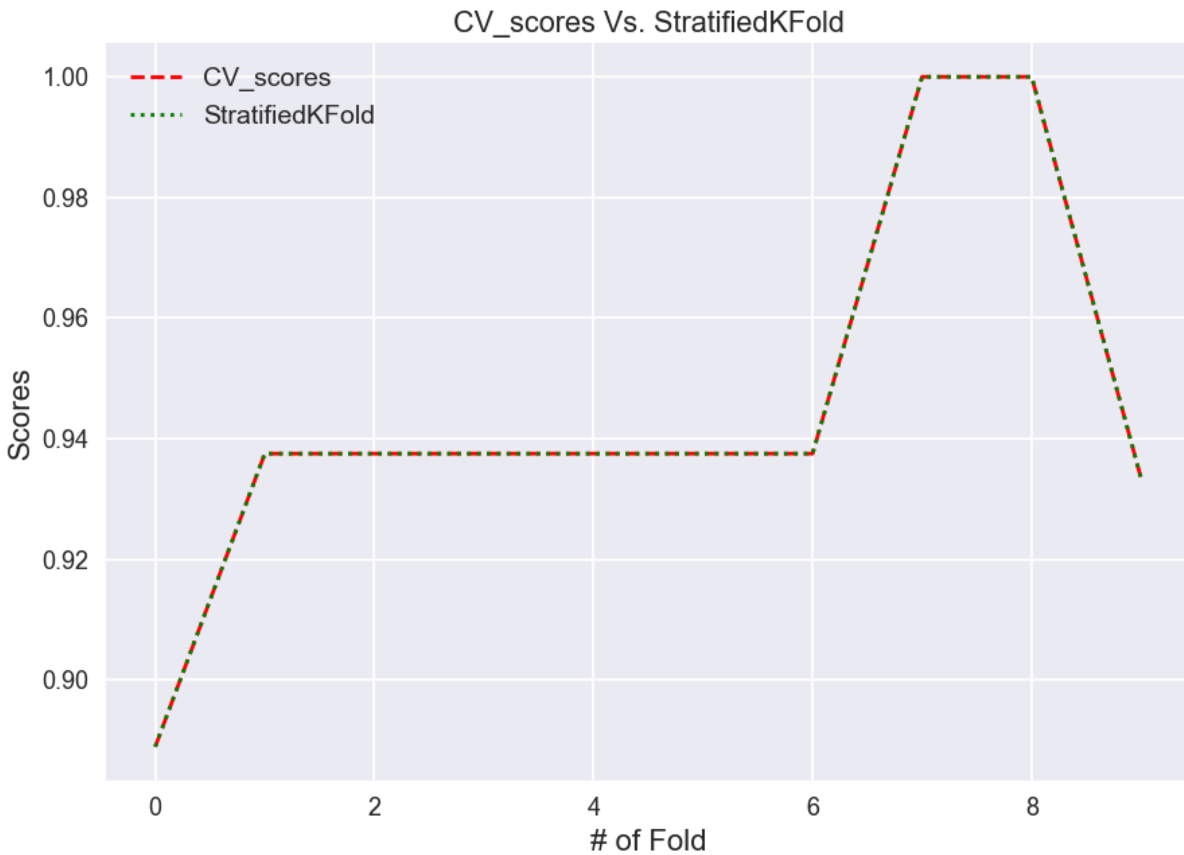
1) Split dataset into ratio 7:3 for training and test sets, respectively. Then use pipeline with StandardScaler(), PCA (n=3), and SVM with RBF kernel to fit the training set and predict the test set. Report the accuracy score.

2) Use StratifiedKFold cross-validation to report the accuracy score (mean with std). What are differences between standard k-fold and StratifiedKFold? What are the differences between StratifiedKFold and cross_val_score (in [12] and [13] of this notebook)?

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import random
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC

df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/'
                      'machine-learning-databases/wine/wine.data',
                      header=None)

print(df_wine.shape)
from sklearn.preprocessing import LabelEncoder
X = df_wine.loc[:, 1:].values
y = df_wine.loc[:, 0].values
le = LabelEncoder()
y = le.fit_transform(y)
print(le.classes_)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=0)
pipe_lr = make_pipeline(StandardScaler(),PCA(n_components=3), SVC(random_state=1))
pipe_lr.fit(X_train, y_train)
y_pred = pipe_lr.predict(X_test)
print('Test Accuracy: %.3f' % pipe_lr.score(X_test, y_test))

from sklearn.model_selection import KFold
from sklearn import svm
svc = svm.SVC(C=1, kernel='linear')
kf = KFold(n_splits=10)
kf.get_n_splits(X)
# print(kf)
KF_scores = list()
for train_index, test_index in kf.split(X):
    # print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    KF_scores.append(svc.fit(X_train, y_train).score(X_test, y_test))
print('\nKF_scores: %s' % KF_scores)
print('KF accuracy: %.3f +/- %.3f' % (np.mean(KF_scores), np.std(KF_scores)))

from sklearn.model_selection import StratifiedKFold
kfold = StratifiedKFold(n_splits=10,random_state=1).split(X_train, y_train)
StratifiedKFold_scores = []
for k, (train, test) in enumerate(kfold):
    pipe_lr.fit(X_train[train], y_train[train])
    score = pipe_lr.score(X_train[test], y_train[test])
    StratifiedKFold_scores.append(score)
    print('StratifiedFold: %2d, Class dist.: %s, Acc: %.3f' % (k + 1,np.bincount(y_train[train]), score))
print('StratifiedKFold CV accuracy: %.3f +/- %.3f' % (np.mean(StratifiedKFold_scores), np.std(StratifiedKFold_scores)))

from sklearn.model_selection import cross_val_score
CV_scores = cross_val_score(estimator=pipe_lr, X=X_train,y=y_train, cv=10, n_jobs=1)
print('\nCV accuracy scores: %s' % CV_scores)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(CV_scores), np.std(CV_scores)))

sns.set()
sns.set_context("talk")
plt.plot(KF_scores,color='r',lw = 2.0, linestyle='--',label='standard k-fold')
plt.plot(StratifiedKFold_scores,color='g', lw = 2.0, linestyle=':',label='StratifiedKFold')
plt.ylabel('Scores', fontsize = 16)
plt.xlabel('# of Fold', fontsize = 16)
plt.title("standard k-fold Vs. StratifiedKFold")
plt.legend(loc='best',fontsize = 14)
plt.show()


plt.plot(CV_scores,color='r',lw = 2.0, linestyle='--',label='CV_scores')
plt.plot(StratifiedKFold_scores,color='g', lw = 2.0, linestyle=':',label='StratifiedKFold')
plt.ylabel('Scores', fontsize = 16)
plt.xlabel('# of Fold', fontsize = 16)
plt.title("CV_scores Vs. StratifiedKFold")
plt.legend(loc='best',fontsize = 14)
```

```
plt.show()

Output:
(178, 14)
[1 2 3]
Test Accuracy: 0.963

KF_scores: [1.0, 0.9444444444444442, 1.0, 0.7777777777777779, 0.8888888888888884, 0.9444444444444442, 1.0, 0.8888888888888884, 1.0, 1.0]
KF accuracy: 0.944 +/- 0.070
StratifiedFold:  1, Class dist.: [53 63 27], Acc: 0.889
StratifiedFold:  2, Class dist.: [53 64 28], Acc: 0.938
StratifiedFold:  3, Class dist.: [53 64 28], Acc: 0.938
StratifiedFold:  4, Class dist.: [53 64 28], Acc: 0.938
StratifiedFold:  5, Class dist.: [53 64 28], Acc: 0.938
StratifiedFold:  6, Class dist.: [53 64 28], Acc: 0.938
StratifiedFold:  7, Class dist.: [53 64 28], Acc: 0.938
StratifiedFold:  8, Class dist.: [53 64 28], Acc: 1.000
StratifiedFold:  9, Class dist.: [53 64 28], Acc: 1.000
StratifiedFold: 10, Class dist.: [54 64 28], Acc: 0.933
StratifiedKFold CV accuracy: 0.945 +/- 0.031

CV accuracy scores: [ 0.88888889  0.9375      0.9375      0.9375      0.9375      0.9375
  0.9375      1.          1.          0.93333333]
CV accuracy: 0.945 +/- 0.031
```

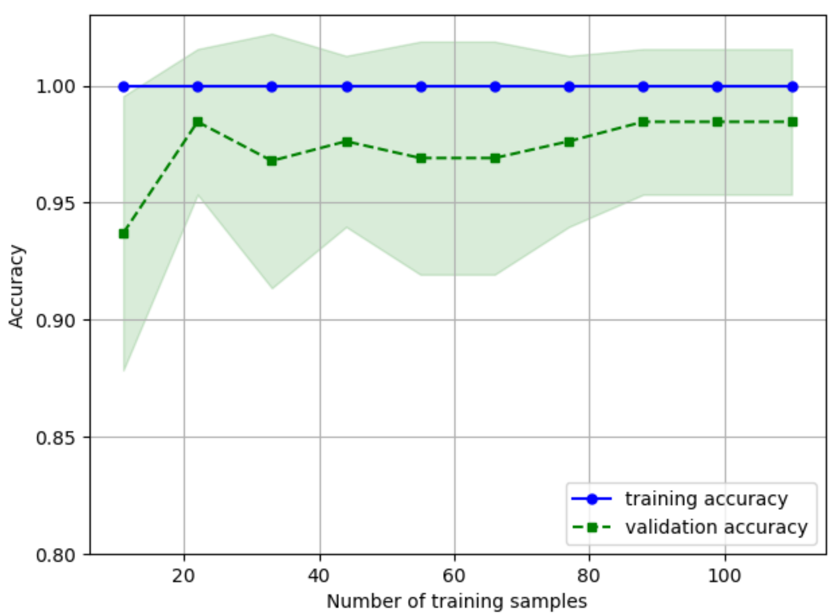


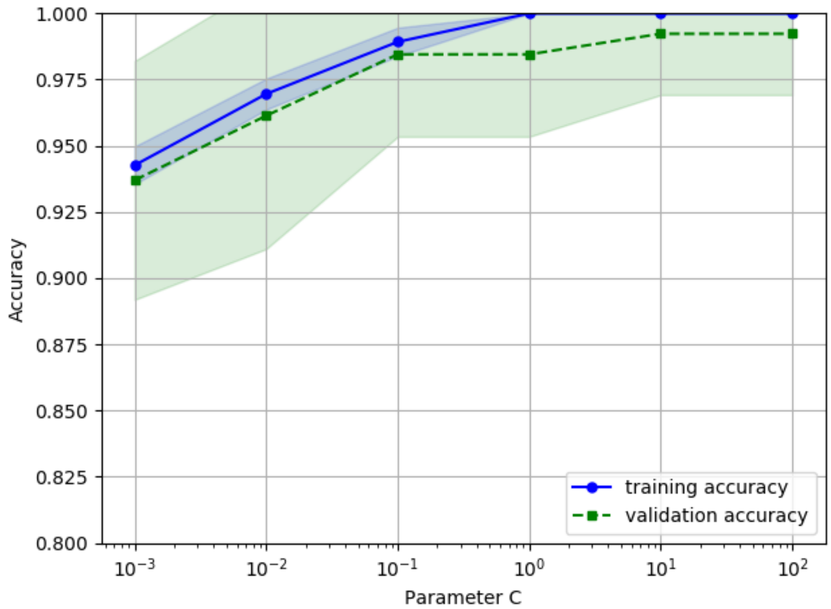


3) What are the differences between "learning curve" and "validation curve" tools in sklearns.model_selection? Report the figures of learning curve and validation curve similar to input [15] and [16] of this notebook, respectively. Based on the figure, indicate which is the best value C you should choose.

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline

pipe_lr = make_pipeline(StandardScaler(),
                        LogisticRegression(penalty='l2', random_state=1))

train_sizes, train_scores, test_scores =\
                learning_curve(estimator=pipe_lr,
                               X=X_train,
                               y=y_train,
                               train_sizes=np.linspace(0.1, 1.0, 10),
                               cv=10,
                               n_jobs=1)

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.plot(train_sizes, train_mean,
         color='blue', marker='o',
         markersize=5, label='training accuracy')

plt.fill_between(train_sizes,
                 train_mean + train_std,
                 train_mean - train_std,
                 alpha=0.15, color='blue')

plt.plot(train_sizes, test_mean,
         color='green', linestyle='--',
         marker='s', markersize=5,
         label='validation accuracy')

plt.fill_between(train_sizes,
                 test_mean + test_std,
                 test_mean - test_std,
```

```
                 alpha=0.15, color='green')

plt.grid()
plt.xlabel('Number of training samples')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.ylim([0.8, 1.03])
plt.tight_layout()
#plt.savefig('images/06_05.png', dpi=300)
plt.show()


from sklearn.model_selection import validation_curve
param_range = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]
train_scores, test_scores = validation_curve(
                estimator=pipe_lr,
                X=X_train,
                y=y_train,
                param_name='logisticregression__C',
                param_range=param_range,
                cv=10)

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.plot(param_range, train_mean,
         color='blue', marker='o',
         markersize=5, label='training accuracy')

plt.fill_between(param_range, train_mean + train_std,
                 train_mean - train_std, alpha=0.15,
                 color='blue')

plt.plot(param_range, test_mean,
         color='green', linestyle='--',
         marker='s', markersize=5,
         label='validation accuracy')

plt.fill_between(param_range,
                 test_mean + test_std,
                 test_mean - test_std,
                 alpha=0.15, color='green')

plt.grid()
plt.xscale('log')
plt.legend(loc='lower right')
plt.xlabel('Parameter C')
plt.ylabel('Accuracy')
plt.ylim([0.8, 1.0])
plt.tight_layout()
plt.show()
```

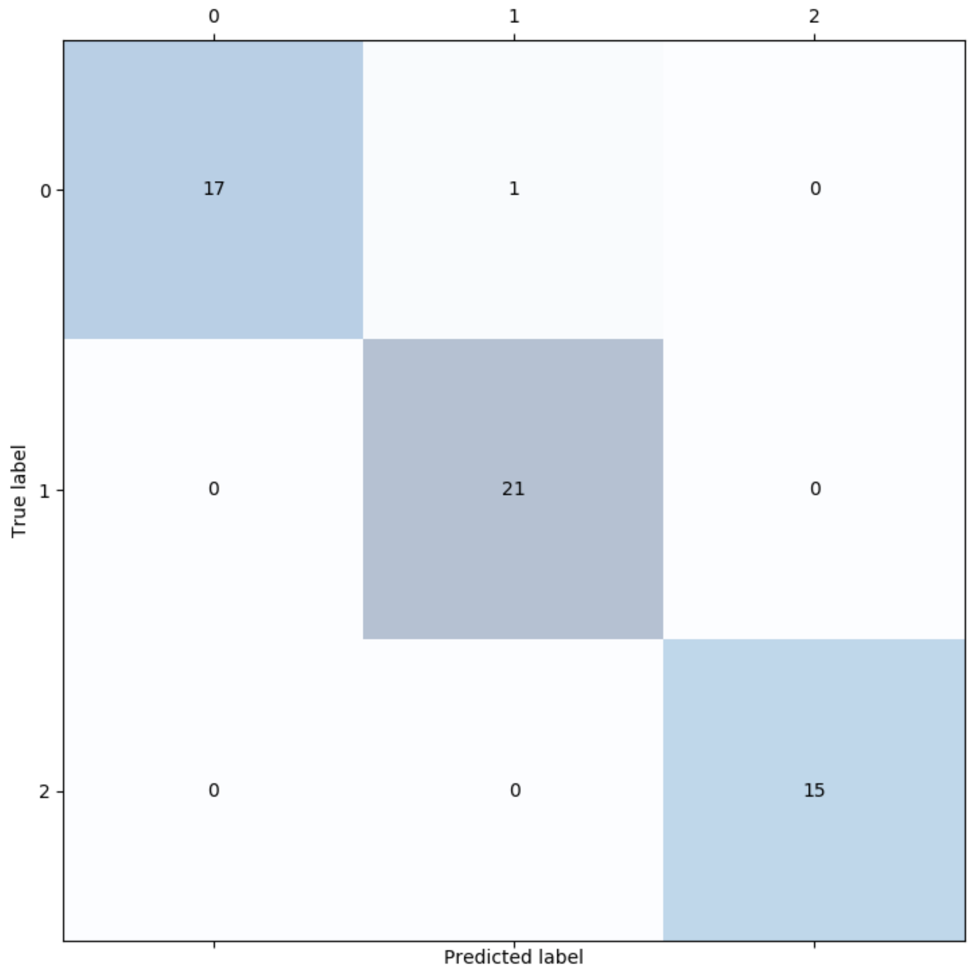




4) Using GridSearchCV to find the best hyperparameter (similar to [17] of this notebook). Compare the accuracy score using these GridSearchCV parameters with previous methods.
5) Report the confusion matrix of the above prediction model using GridSearchCV.
6) Report the precision and recall scores as in [29] and the best scores and best parameter of GridSearchCV as in [30] of this notebook.

```
from sklearn.model_selection import GridSearchCV
    from sklearn.svm import SVC

    pipe_svc = make_pipeline(StandardScaler(),
                             SVC(random_state=1))

    param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]

    param_grid = [{'svc__C': param_range,
                  'svc__kernel': ['linear']},
                 {'svc__C': param_range,
                  'svc__gamma': param_range,
                  'svc__kernel': ['rbf']}]

if __name__ == '__main__':
    gs = GridSearchCV(estimator=pipe_svc,
                      param_grid=param_grid,
                      scoring='accuracy',
                      cv=10,
                      n_jobs=-1)
    gs = gs.fit(X_train, y_train)
    print(gs.best_score_)
    print(gs.best_params_)

    clf = gs.best_estimator_
    clf.fit(X_train, y_train)
    print('Test accuracy: %.3f' % clf.score(X_test, y_test))

    gs = GridSearchCV(estimator=pipe_svc,
                      param_grid=param_grid,
                      scoring='accuracy',
                      cv=2)

    scores = cross_val_score(gs, X_train, y_train,
                             scoring='accuracy', cv=10)
    print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),
                                          np.std(scores)))


from sklearn.metrics import confusion_matrix

    pipe_svc.fit(X_train, y_train)
    y_pred = pipe_svc.predict(X_test)
    confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)
    print(confmat)

    fig, ax = plt.subplots(figsize=(2.5, 2.5))
    ax.matshow(confmat, cmap=plt.cm.Blues, alpha=0.3)
    for i in range(confmat.shape[0]):
        for j in range(confmat.shape[1]):
            ax.text(x=j, y=i, s=confmat[i, j], va='center', ha='center')
```

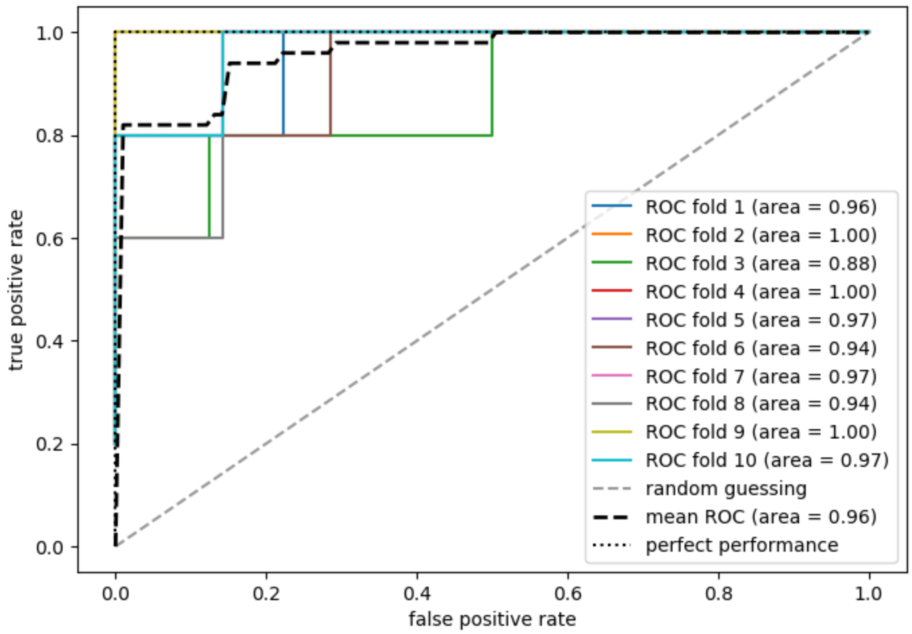
```
    plt.xlabel('Predicted label')
    plt.ylabel('True label')

    plt.tight_layout()
    plt.show()


from sklearn.metrics import precision_score, recall_score, f1_score
    # will return the total ratio of tp/(tp + fp)
    print('Precision: %.3f' % precision_score(y_true=y_test, y_pred=y_pred, average='micro'))
    print('Recall: %.3f' % recall_score(y_true=y_test, y_pred=y_pred, average='micro'))
    print('F1: %.3f' % f1_score(y_true=y_test, y_pred=y_pred, average='micro'))

    from sklearn.metrics import make_scorer

    scorer = make_scorer(f1_score, pos_label=0)

    c_gamma_range = [0.01, 0.1, 1.0, 10.0]

    param_grid = [{'svc__C': c_gamma_range,
                   'svc__kernel': ['linear']},
                  {'svc__C': c_gamma_range,
                   'svc__gamma': c_gamma_range,
                   'svc__kernel': ['rbf']}]

    gs = GridSearchCV(estimator=pipe_svc,
                      param_grid=param_grid,
                      scoring=scorer,
                      cv=10,
                      n_jobs=-1)
    gs = gs.fit(X_train, y_train)
    print(gs.best_score_)
    print(gs.best_params_)

Output:
0.991935483871
{'svc__C': 0.1, 'svc__kernel': 'linear'}
Test accuracy: 1.000
CV accuracy: 0.976 +/- 0.037
[[17  1  0]
 [ 0 21  0]
 [ 0  0 15]]
Precision: 0.981
Recall: 0.981
F1: 0.981
```



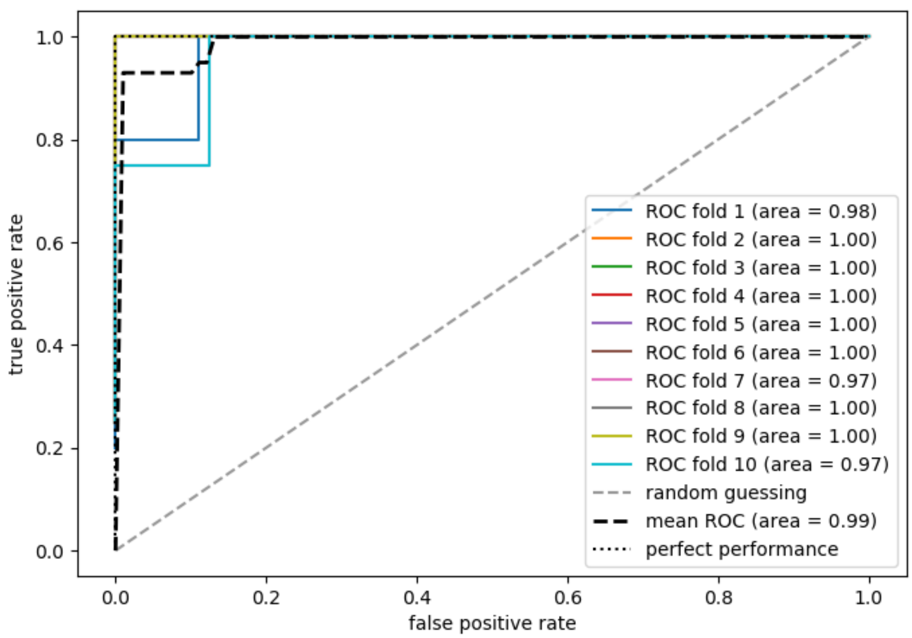
7) Plotting the ROCs for every pair combination of classes as in [31] of this notebook, or use the 3 dimension ROCs if it is available.

```
pipe_lr = make_pipeline(StandardScaler(),
                        PCA(n_components=3),
                        LogisticRegression(penalty='l2',
                                           random_state=1,
                                           C=100.0))
X_train2 = X_train[:, 1:]
cv = list(StratifiedKFold(n_splits=10,
                          random_state=1).split(X_train, y_train))

fig = plt.figure(figsize=(7, 5))

mean_tpr = 0.0
mean_fpr = np.linspace(0, 1, 100)
all_tpr = []

for i, (train, test) in enumerate(cv):
    probas = pipe_lr.fit(X_train2[train],
                         y_train[train]).predict_proba(X_train2[test])

    fpr, tpr, thresholds = roc_curve(y_train[test],
                                     probas[:, 0],
                                     pos_label=1)
    mean_tpr += interp(mean_fpr, fpr, tpr)
    mean_tpr[0] = 0.0
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr,
             tpr,
             label='ROC fold %d (area = %0.2f)'
                   % (i + 1, roc_auc))

plt.plot([0, 1],
         [0, 1],
         linestyle='--',
         color=(0.6, 0.6, 0.6),
         label='random guessing')

mean_tpr /= len(cv)
```

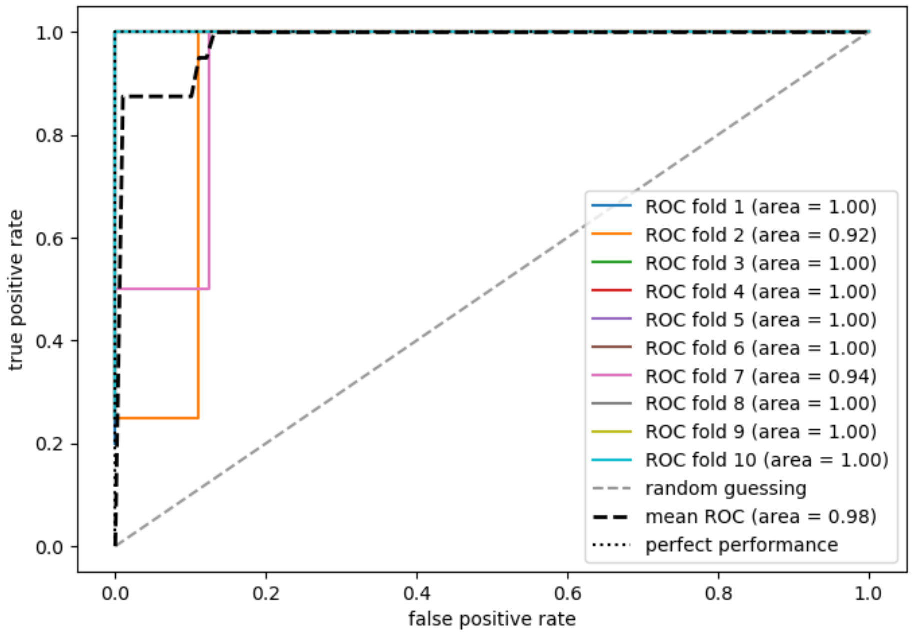
```
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
plt.plot(mean_fpr, mean_tpr, 'k--',
         label='mean ROC (area = %0.2f)' % mean_auc, lw=2)
plt.plot([0, 0, 1],
         [0, 1, 1],
         linestyle=':',
         color='black',
         label='perfect performance')

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.legend(loc="lower right")

plt.tight_layout()
plt.show()
```



PCA = 2



PCA = 3



hw3

This private post is only visible to Instructors and MD SHIRAJUM MUNIR

Updated 2 years ago by MD SHIRAJUM MUNIR

followup discussions *for lingering questions and comments*