Problem3_Questions0816

September 7, 2020

1 Instructions

1. If there is a conflict bewteen the problem description in the ipython notebook and the question in the pdf, follow the question in the pdf file.
2. The part you need to fill in is commented as "Code Clip". You can search "Code Clip" in this notebook to find the part you need to complete. After you finish the required part, you may need to run other related code blocks for evaluation or visualization.
3. If you have a better implementation or find mistakes in this notebook, you could add/modify any function (input, output and return) yourself. Everything is flexible as long as you answered the questions.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.metrics import accuracy_score, auc, roc_curve
from sklearn import preprocessing
from sklearn import metrics
from scipy.stats import ttest_rel
```

[3]:
```
# set random seed for reproducibility
np.random.seed(1000)
```

1.0.1 Load Training and Testing Data. Get a initial statistics of the training data.

[4]:
```
train_data = pd.read_csv('./data_train.csv')
test_data = pd.read_csv('./data_test.csv')
```

[5]:
```
features_mean = list(train_data.columns[1:31])

X_train = train_data.loc[:,features_mean]
y_train = train_data.loc[:, 'diagnosis']

X_test = test_data.loc[:,features_mean]
y_test = test_data.loc[:, 'diagnosis']
```

2 3.1 Balanced Dataset

2.0.1 3.1.1 Use 5-fold cross validation on the training set only, and compare accuracy and time cost performance of three different algorithms: ID3, CART and Random Forest,

Code Clip 3.1.1a: Complete the function `compare`.

[6]:
```
from sklearn.metrics import confusion_matrix
def accuracy_per_class(predict, label):
    cm = confusion_matrix(label, predict)
    return np.diag(cm)/np.sum(cm, axis = 1)
```

[7]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
import time
start = time.time()


def eval_tree(model_class, X_train, y_train, X_test, y_test, folds,␣
↪class_weight, max_depth, **model_args):
    model = model_class(**model_args, class_weight=class_weight,␣
↪max_depth=max_depth)

    # train on entire training set and evaluate on test set
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    test_accuracy = accuracy_score(y_test, y_pred)

    # accuracy per class
    class_accuracy = accuracy_per_class(y_pred, y_test)

    # 5-fold cross validation
    val_scores = []
    times = []

    for train_index, val_index in folds:
        # split data into training and validation
        X_t, X_v = X_train.loc[train_index], X_train.loc[val_index]
        y_t, y_v = y_train.loc[train_index], y_train.loc[val_index]

        start = time.time()

        # fit model and find validation score
        model = model_class(**model_args, class_weight=class_weight,␣
↪max_depth=max_depth)
        model.fit(X_t, y_t)
```

```
        fold_accuracy = accuracy_score(y_v, model.predict(X_v))

        # record the time taken to fit the model
        time_taken = time.time() - start

        # save the validation score and time taken
        val_scores.append(fold_accuracy)
        times.append(time_taken)

    return test_accuracy, class_accuracy, np.array(val_scores), np.array(times)


def compare(X_train, y_train, X_test, y_test, class_weight = None, max_depth =␣
↪None):
    data = []

    # Code Clip 3.1.1a

    # split training data into folds beforehand so that all methods use the␣
↪same folds for cross-validation
    kf = KFold(n_splits=5)
    folds = list(kf.split(X_train))

    #-------------Forest-------------
    d = eval_tree(RandomForestClassifier, X_train, y_train, X_test, y_test,␣
↪folds, class_weight, max_depth, n_estimators=50)
    data.append(d)

    #-------------CART-------------
    d = eval_tree(DecisionTreeClassifier, X_train, y_train, X_test, y_test,␣
↪folds, class_weight, max_depth, criterion='gini')
    data.append(d)

    #-------------ID3-------------
    d = eval_tree(DecisionTreeClassifier, X_train, y_train, X_test, y_test,␣
↪folds, class_weight, max_depth, criterion='entropy')
    data.append(d)

    # return test accuracies, class accuracies, validation scores, and times
    return zip(*data)
```

[8]:
```
test_accuracies, class_accuracies, val_scores, times = compare(X_train,␣
↪y_train, X_test, y_test)
```

Code Clip 3.1.1b: Do pairwise t-tests to determine whether there's a significant difference between the best algorithm and the other algorithms. (Hint: You could first use `sklearn.model_selection.KFold` to get 5 different train/val division on the training set.)

[9]:
```
# Code Clip 3.1.1b
```

[10]:
```
def significant(*vals, p=0.05):
    '''
    Returns true if the best distribution is statistically significantly better␣
 ↪than the others, False otherwise
    ----------
    *vals: each 'val' should be an array of validation scores computed with␣
 ↪n-fold cross validation
    '''
    # find distribution with highest mean
    means = [v.mean() for v in vals]
    i_max = np.argmax(means)

    # check if other distributions are significantly different
    for i in range(len(vals)):
        if i != i_max:
            # perform t-test
            _, p_stat = ttest_rel(vals[i_max], vals[i])
            if p_stat > p:
                return False
    return True


names = ['Random Forest', 'CART', 'ID3']
for i in range(len(names)):
    print(names[i])
    print('-' * 10)
    print(f'Test accuracy: {round(test_accuracies[i], 4)}')
    print(f'Class accuracies: {round(class_accuracies[i][0], 4)},␣
 ↪{round(class_accuracies[i][1], 4)}')
    print(f'Validation scores: {round(val_scores[i].mean(), 4)} +-␣
 ↪{round(val_scores[i].std(), 4)}')
    print(f'Validation Times: {round(times[i].mean(), 4)} +- {round(times[i].
 ↪std(), 4)}')
    print()

print('=' * 20)

print('\nSignificance Tests')
print('-' * 10)
print(f'Best method is better: {significant(*val_scores)}')
print(f'Best method takes longer: {significant(*times)}')
```

```
Random Forest
----------
Test accuracy: 0.9649
```

```
Class accuracies: 0.9859, 0.9302
Validation scores: 0.9604 +- 0.0088
Validation Times: 0.0766 +- 0.0043

CART
----------
Test accuracy: 0.9386
Class accuracies: 0.9577, 0.907
Validation scores: 0.9231 +- 0.0197
Validation Times: 0.0065 +- 0.0006

ID3
----------
Test accuracy: 0.9474
Class accuracies: 0.9859, 0.8837
Validation scores: 0.9341 +- 0.0155
Validation Times: 0.0073 +- 0.0002

====================

Significance Tests
----------
Best method is better: False
Best method takes longer: True
```

2.0.2 3.1.2 Effect of the depth.

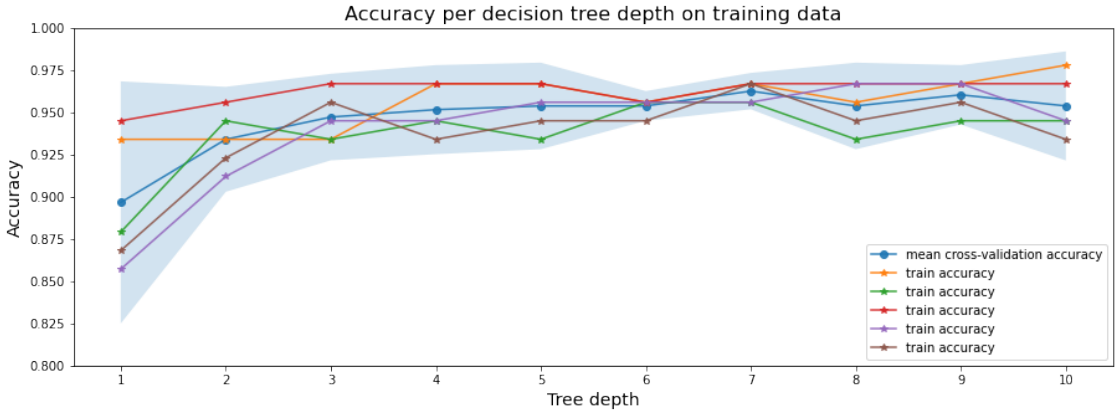
Code Clip 3.1.2: Complete the function `run_cross_validation_on_trees`.

[11]:
```python
def run_cross_validation_on_trees(X, y, tree_depths, cv=5, scoring='accuracy'):
    cv_scores = []        # lists of CV accuracies for each tree depth
    cv_means = []         # means of CV accuracies for each tree depth
    cv_stds = []          # standard deviations of CV accuracies for each tree␣
 ↪depth

    # calculate fold indices beforehand so each depth is evaluated with the␣
 ↪same folds
    kf = KFold(n_splits=5)
    folds = list(kf.split(X))

    # loop through tree depths
    for depth in tree_depths:

        acc = []

        # loop through folds
        for train_index, val_index in folds:
```

```
            # split data into training and validation
            X_t, X_v = X.loc[train_index], X.loc[val_index]
            y_t, y_v = y.loc[train_index], y.loc[val_index]

            # train and evaluate model
            model = RandomForestClassifier(n_estimators=50, max_depth=depth)
            model.fit(X_t, y_t)
            fold_accuracy = accuracy_score(y_v, model.predict(X_v))

            # record fold accuracy
            acc.append(fold_accuracy)

        # record accuracy statistics
        acc = np.array(acc)
        cv_scores.append(acc)
        cv_means.append(acc.mean())
        cv_stds.append(acc.std())

    return np.array(cv_scores), np.array(cv_means), np.array(cv_stds)

# function for plotting cross-validation results
def plot_cross_validation_on_trees(depths, cv_scores_mean, cv_scores_std,␣
↪accuracy_scores, title):
    fig, ax = plt.subplots(1,1, figsize=(15,5))
    ax.plot(depths, cv_scores_mean, '-o', label='mean cross-validation␣
↪accuracy', alpha=0.9)
    ax.fill_between(depths, cv_scores_mean-2*cv_scores_std,␣
↪cv_scores_mean+2*cv_scores_std, alpha=0.2)
    ylim = plt.ylim()
    ax.plot(depths, accuracy_scores, '-*', label='train accuracy', alpha=0.9)
    ax.set_title(title, fontsize=16)
    ax.set_xlabel('Tree depth', fontsize=14)
    ax.set_ylabel('Accuracy', fontsize=14)
    ax.set_ylim([0.8,1])
    ax.set_xticks(depths)
    ax.legend()
```

[12]:
```
sm_tree_depths = range(1,11)
sm_accuracy_scores, sm_cv_scores_mean, sm_cv_scores_std =␣
↪run_cross_validation_on_trees(X_train, y_train, sm_tree_depths)

# plotting accuracy
plot_cross_validation_on_trees(sm_tree_depths, sm_cv_scores_mean,␣
↪sm_cv_scores_std, sm_accuracy_scores,
                               'Accuracy per decision tree depth on training␣
↪data')
```

```
plt.show()
```



[13]:
```
print('CV accuracies for each tree depth')
print('-' * 10)
for i in range(10):
    print(f'Depth {i + 1}: {round(sm_cv_scores_mean[i], 4)} +-␣
 ↪{round(sm_cv_scores_std[i], 4)}')
```

```
CV accuracies for each tree depth
----------
Depth 1: 0.8967 +- 0.0358
Depth 2: 0.9341 +- 0.0155
Depth 3: 0.9473 +- 0.0128
Depth 4: 0.9516 +- 0.0132
Depth 5: 0.9538 +- 0.0128
Depth 6: 0.9538 +- 0.0044
Depth 7: 0.9626 +- 0.0054
Depth 8: 0.9538 +- 0.0128
Depth 9: 0.9604 +- 0.0088
Depth 10: 0.9538 +- 0.0162
```

[14]:
```
# train a model on the full training set with depth chosen based on the CV␣
 ↪results
depth = np.argmax(sm_cv_scores_mean) + 1

model = RandomForestClassifier(n_estimators=50, max_depth=depth)
model.fit(X_train, y_train)
accuracy = accuracy_score(y_test, model.predict(X_test))

print(f'Depth chosen: {depth}')
print(f'Accuracy on test set: {round(accuracy, 4)}')
```

```
Depth chosen: 7
```

```
Accuracy on test set: 0.9649
```

2.0.3 3.1.3 ROC and vairable importance

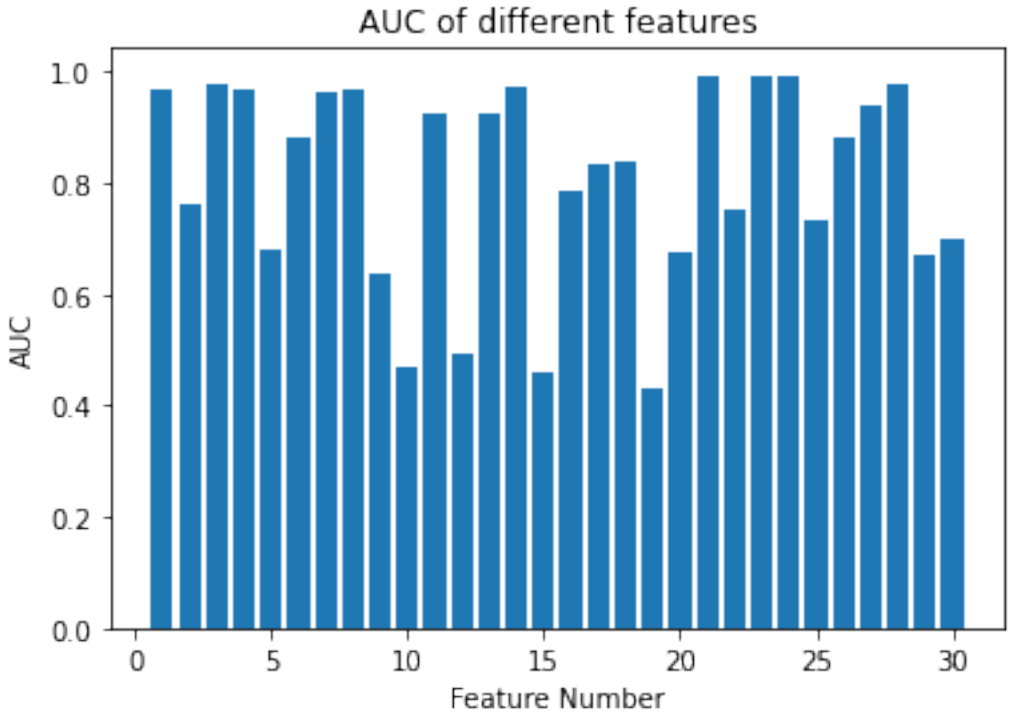
Code Clip 3.1.3a: Complete the function `draw_roc_with_feature_idx`. Then finished the next two steps (`AUC of different features` and `Draw ROC Curve of the first five features`.

[15]:
```
def plot_roc(fpr, tpr, roc_auc, title=None, legend=True):
    plt.figure()
    lw = 2
    try:
        fpr[0][0]
        for i in range(len(fpr)):
            plt.plot(fpr[i], tpr[i], color='darkorange',
                     lw=lw, label=f'Feature {i+1} AUC ={round(roc_auc[i], 4)}')
    except IndexError:
        plt.plot(fpr, tpr, color='darkorange',
                     lw=lw, label='ROC curve (area ={})'.format(round(roc_auc,␣
 ↪4)))
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic over {}'.format(title))
    if legend:
        plt.legend(loc="lower right")
    plt.show()


def draw_roc_with_feature_idx(X_test, y_test, i, draw = False):

    # Code Clip 3.1.3
    # calculate list of false positive rate and true positive rate.
    # calculate AUC.

    fpr, tpr, thresholds = roc_curve(y_test, X_test[features_mean[i]])
    roc_auc = auc(fpr, tpr)


    if draw:
        plot_roc(fpr, tpr, roc_auc, title = 'Feature' + str(i))

    return fpr, tpr, roc_auc
```

2.0.4 AUC of different features

Code clip 3.1.3b, You may find plt.bar useful here.

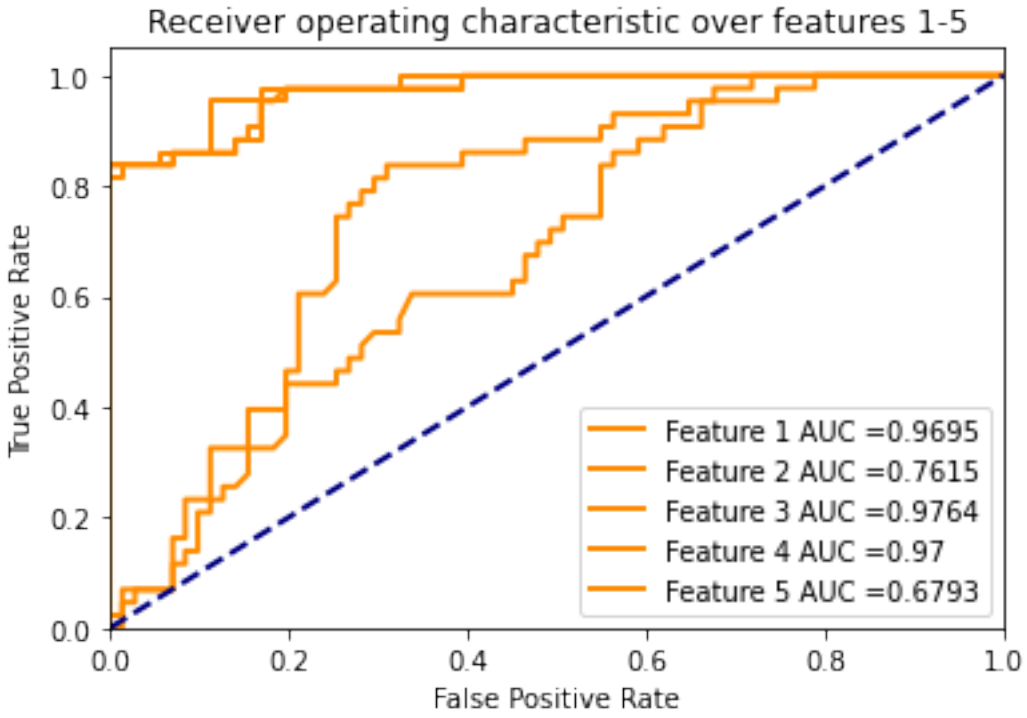
[16]:
```
plt.figure()

# Code Clip 3.1.3b
features = range(1, 31)
roc_aucs = [draw_roc_with_feature_idx(X_test, y_test, i - 1)[-1] for i in␣
 ↪features]
plt.bar(features, roc_aucs)

plt.xlabel('Feature Number')
plt.ylabel('AUC')
plt.title('AUC of different features')
plt.show()
```

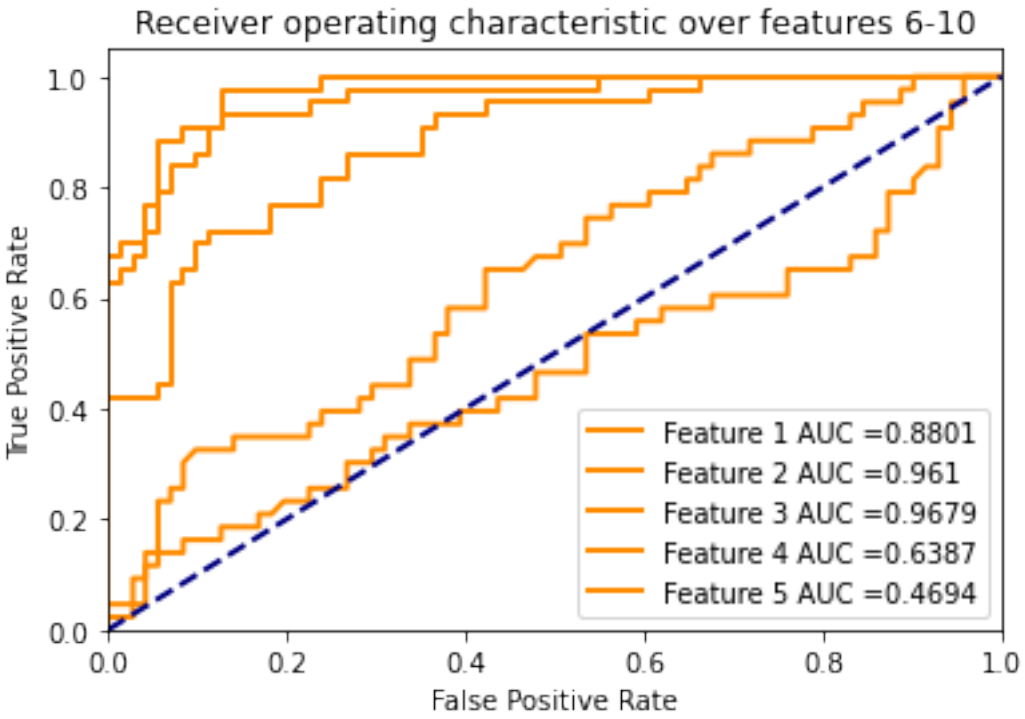


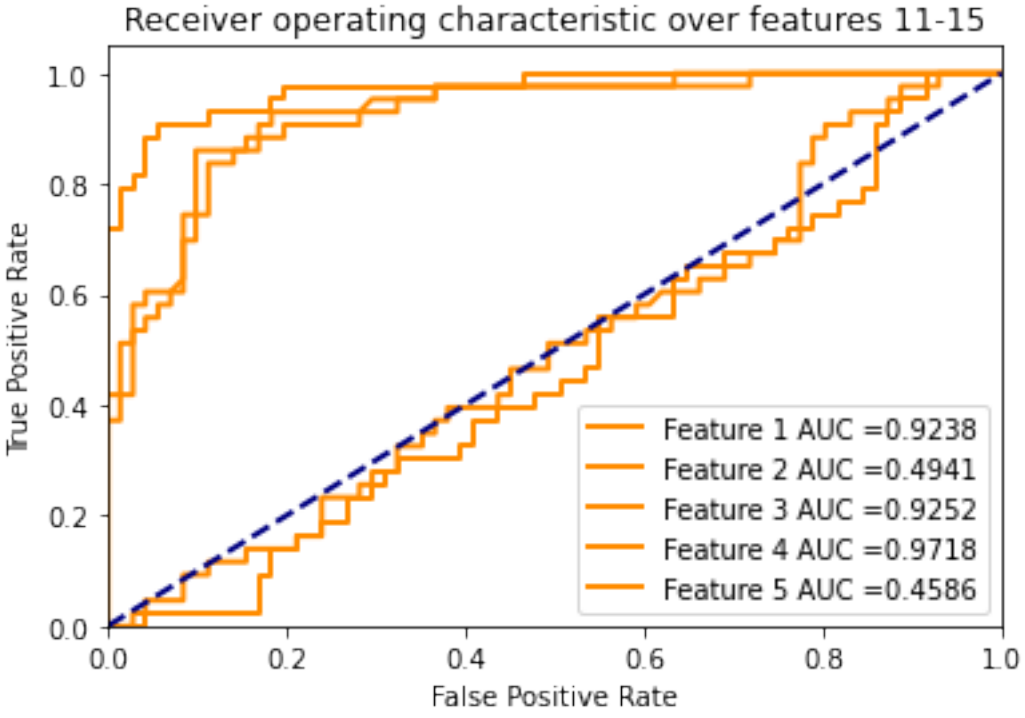
2.0.5 Draw ROC Curves for the features

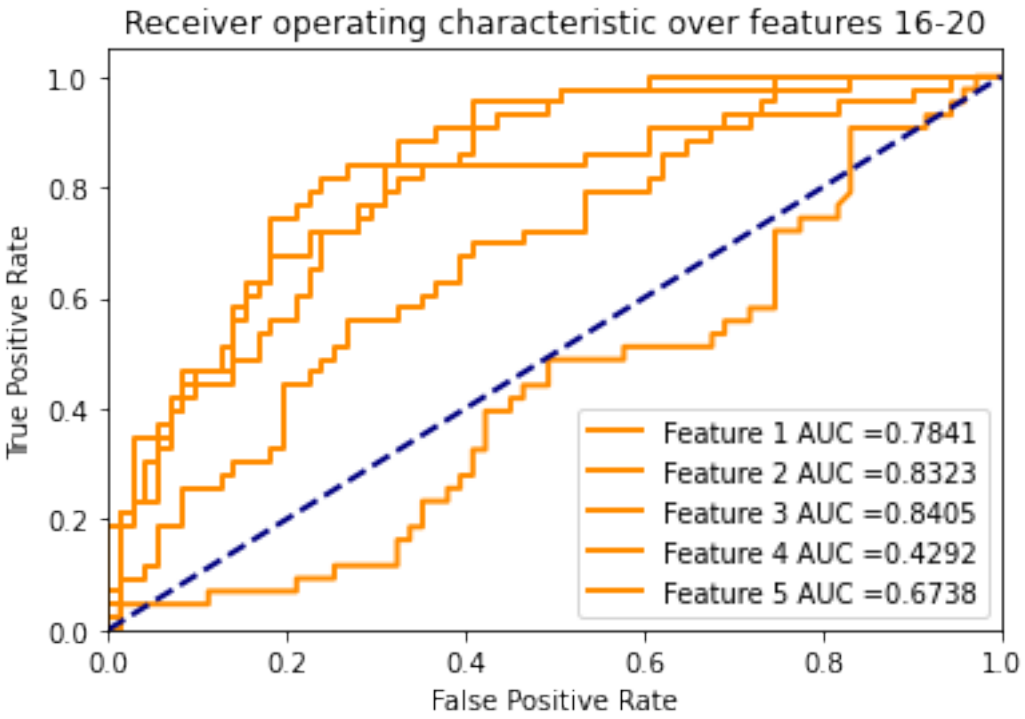
[17]:
```
# draw ROC curves in groups of 5
for i in range(6):
    fpr = []
    tpr = []
    roc_auc = []
    for j in range(5):
        f, t, r = draw_roc_with_feature_idx(X_test, y_test, i*5 + j)
        fpr.append(f)
```

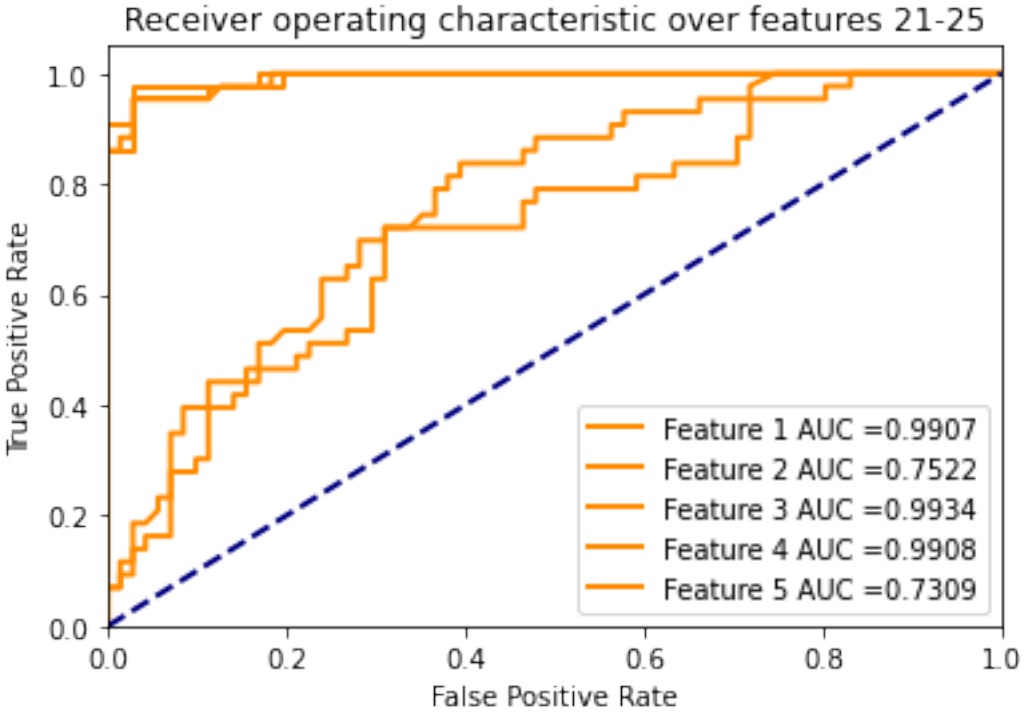
```
        tpr.append(t)
        roc_auc.append(r)

    plot_roc(fpr, tpr, roc_auc, title=f'features {i*5 + 1}-{i*5 + 5}')

# draw ROC curves all on one graph
data = [draw_roc_with_feature_idx(X_test, y_test, i) for i in range(30)]
fpr, tpr, roc_auc = zip(*data)
plot_roc(fpr, tpr, roc_auc, title=f'all features', legend=False)
```

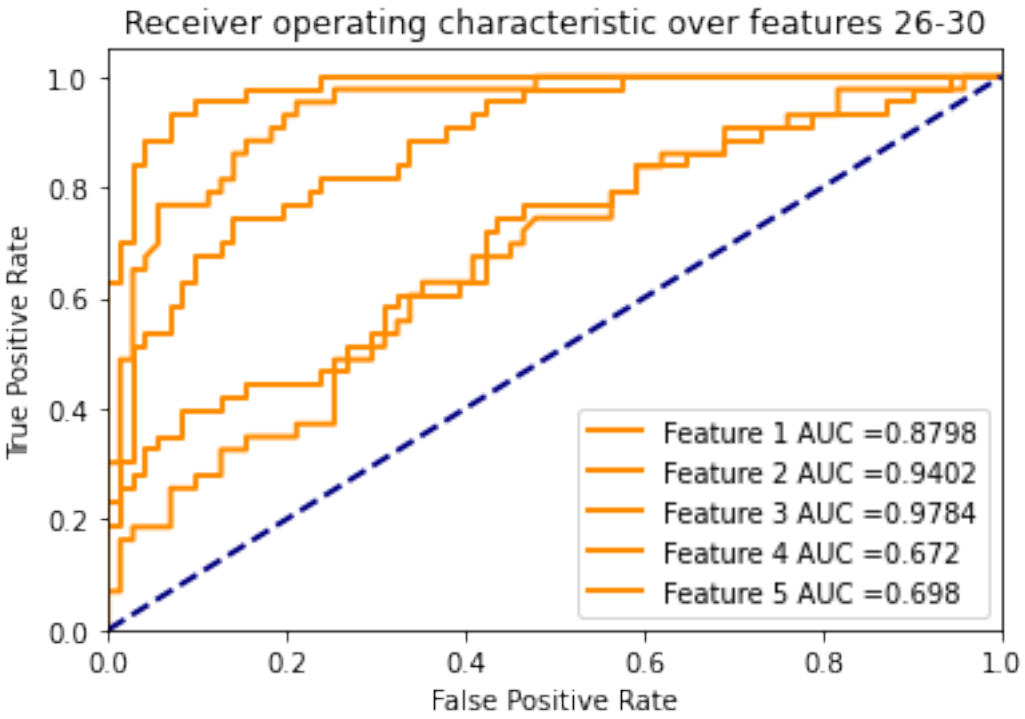


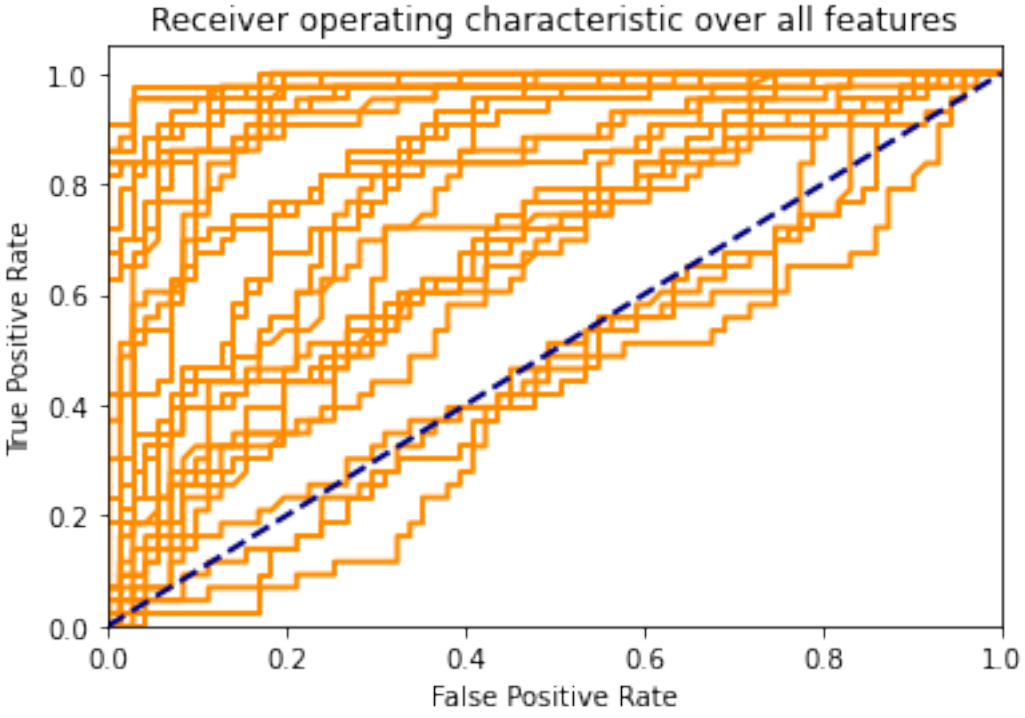

Receiver operating characteristic over features 6-10
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Feature 1 AUC =0.8801
Feature 2 AUC =0.961
Feature 3 AUC =0.9679
Feature 4 AUC =0.6387
Feature 5 AUC =0.4694


Receiver operating characteristic over features 11-15
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Feature 1 AUC =0.9238
Feature 2 AUC =0.4941
Feature 3 AUC =0.9252
Feature 4 AUC =0.9718
Feature 5 AUC =0.4586


Receiver operating characteristic over features 16-20
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Feature 1 AUC =0.7841
Feature 2 AUC =0.8323
Feature 3 AUC =0.8405
Feature 4 AUC =0.4292
Feature 5 AUC =0.6738


Receiver operating characteristic over features 21-25
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Feature 1 AUC =0.9907
Feature 2 AUC =0.7522
Feature 3 AUC =0.9934
Feature 4 AUC =0.9908
Feature 5 AUC =0.7309


Receiver operating characteristic over features 26-30
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Feature 1 AUC =0.8798
Feature 2 AUC =0.9402
Feature 3 AUC =0.9784
Feature 4 AUC =0.672
Feature 5 AUC =0.698


Receiver operating characteristic over all features
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

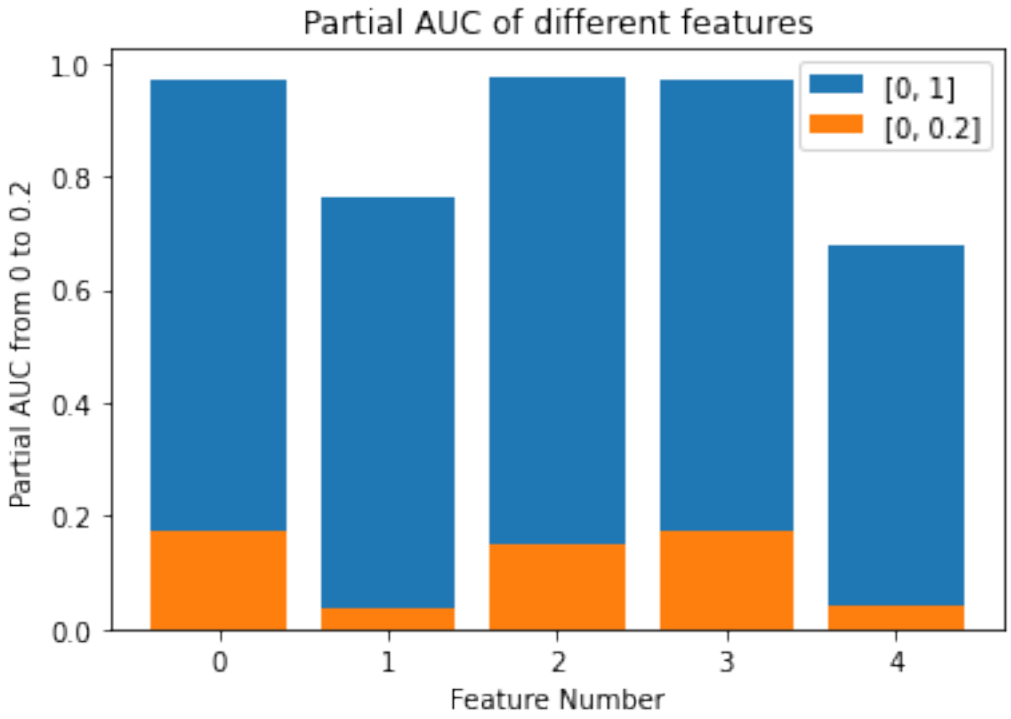
2.0.6 3.1.4 Partial ROC

Code Clip 3.1.4 Follow the intruction in the question. You could test the correctness of your code by setting $t_0 = 0, t_1 = 1$.

[18]:
```
# Code Clip 3.1.4
def partial_roc_auc(X_test, y_test, i, t_0, t_1):
    '''
    Calculates the AUC of an ROC curve based on feature i along the FPR range␣
 ↪[t_0, t_1]
    ----------
    X_test, y_test: test features and labels
    i: index of feature we're evaluating
    [t_0, t_1]: false positive rate interval that we're calculating the AUC for
    '''
    # calculate FPR and TPR for the whole range
    fpr, tpr, thresholds = roc_curve(y_test, X_test[features_mean[i]])

    # determine which false positive rates fall within the desired interval
    min_ind = np.where(t_0 <= fpr)[0][0]
    max_ind = np.where(fpr <= t_1)[0][-1]

    # discard data that doesn't fall in the desired interval
    fpr = fpr[min_ind:max_ind + 1]
    tpr = tpr[min_ind:max_ind + 1]

    # return the partial AUC
    return auc(fpr, tpr)
```

[19]:
```
features = range(5)

# AUCs from 0 to 1
full_aucs = [partial_roc_auc(X_test, y_test, i, 0, 1) for i in features]
plt.bar(features, full_aucs, label='[0, 1]')

# AUCs from 0 to 0.2
p_aucs = [partial_roc_auc(X_test, y_test, i, 0, 0.2) for i in features]
plt.bar(features, p_aucs, label='[0, 0.2]')

# plot the results
plt.xlabel('Feature Number')
plt.ylabel('Partial AUC from 0 to 0.2')
plt.title('Partial AUC of different features')
plt.legend(loc="upper right")
plt.show()

# show the actual values for the partial AUCs from 0 to 0.2
for i in range(5):
```

```
    print(f'Feature {i}: {p_aucs[i]}')
```



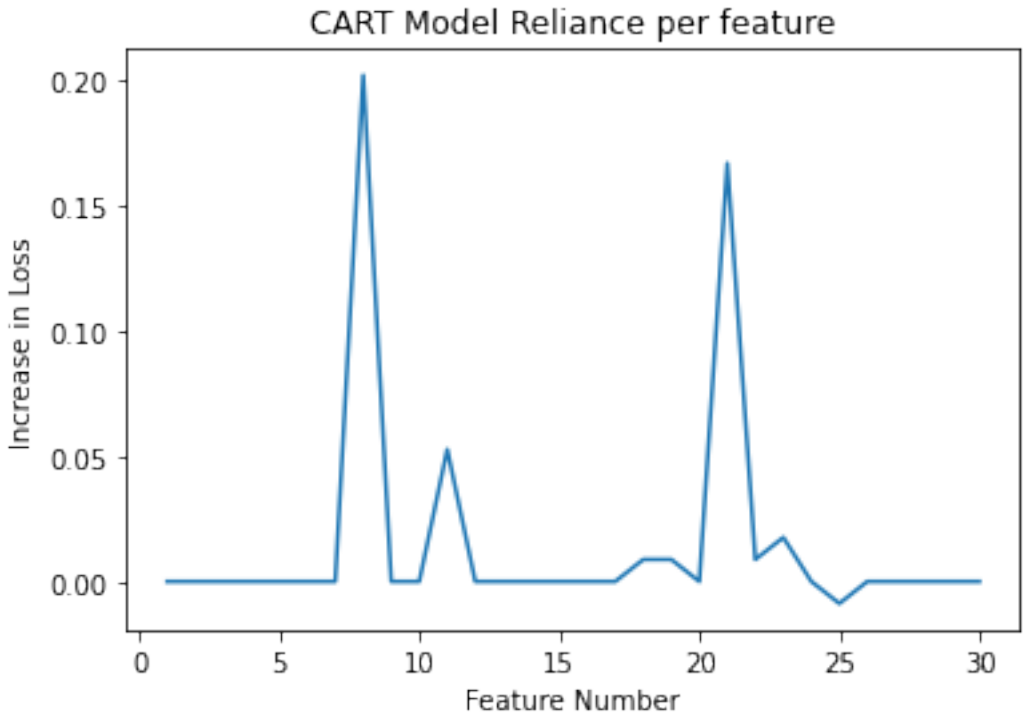

```
Feature 0: 0.17130691123485095
Feature 1: 0.038159187684245
Feature 2: 0.14903373730756633
Feature 3: 0.1717982312479528
Feature 4: 0.040124467736652465
```

2.0.7 3.1.5 Model Reliance of CART model

Code Clip 3.1.5 Follow the instruction in the question.

[20]:
```
# Code Clip 3.1.5
model = DecisionTreeClassifier(criterion='gini')
model.fit(X_train, y_train)

# find base error
error = 1 - accuracy_score(model.predict(X_test), y_test)
model_reliances = []

# find error after scrambling each column of data individually
for idx in range(30):
    f = features_mean[idx]
    X = X_test.copy()
```

```
    X[f] = X[f].sample(n=X.shape[0], replace=False).values
    shuffled_error = 1 - accuracy_score(model.predict(X), y_test)
    model_reliances.append(shuffled_error - error)

# plot results
plt.plot(range(1, 31), model_reliances)

plt.xlabel('Feature Number')
plt.ylabel('Increase in Loss')
plt.title('CART Model Reliance per feature')
plt.show()

# print out numerical results
for i, val in enumerate(model_reliances):
    print(f'Feature {i}: {val}')
```




```
Feature 0: 0.0
Feature 1: 0.0
Feature 2: 0.0
Feature 3: 0.0
Feature 4: 0.0
Feature 5: 0.0
Feature 6: 0.0
Feature 7: 0.20175438596491235
```

```
Feature 8: 0.0
Feature 9: 0.0
Feature 10: 0.052631578947368474
Feature 11: 0.0
Feature 12: 0.0
Feature 13: 0.0
Feature 14: 0.0
Feature 15: 0.0
Feature 16: 0.0
Feature 17: 0.00877192982456143
Feature 18: 0.00877192982456143
Feature 19: 0.0
Feature 20: 0.16666666666666674
Feature 21: 0.00877192982456143
Feature 22: 0.01754385964912286
Feature 23: 0.0
Feature 24: -0.00877192982456132
Feature 25: 0.0
Feature 26: 0.0
Feature 27: 0.0
Feature 28: 0.0
Feature 29: 0.0
```

3 3.2 Imbalanced Dataset

[21]:
```python
train_data = pd.read_csv('./data_imbalanced_train.csv')
test_data = pd.read_csv('./data_imbalanced_test.csv')

features_mean = list(train_data.columns[1:31])

X_train = train_data.loc[:,features_mean]
y_train = train_data.loc[:, 'diagnosis']

X_test = test_data.loc[:,features_mean]
y_test = test_data.loc[:, 'diagnosis']
```

3.0.1 3.2.1 What is the ratio between the two labels ?

Code Clip 3.2.1

[22]:
```python
# Code Clip 3.2.1

# training data
num0_train = len(np.where(y_train == 0)[0])
num1_train = len(np.where(y_train == 1)[0])
print(f'Training data\t|\t0: {num0_train}, 1: {num1_train} \t|\tratio:␣
 ↪{round(num0_train / num1_train, 4)}')
```

```
# testing data
num0_test = len(np.where(y_test == 0)[0])
num1_test = len(np.where(y_test == 1)[0])
print(f'Testing data\t|\t0: {num0_test}, 1: {num1_test} \t|\tratio:␣
 ↪{round(num0_test / num1_test, 4)}')

# overall
print(f'Overall\t\t|\t0: {num0_train + num0_test}, 1: {num1_train + num1_test}␣
 ↪\t|\tratio: {round((num0_test + num0_train) / (num1_train + num1_test), 4)}')
```

```
Training data   |       0: 290, 1: 52   |       ratio: 5.5769
Testing data    |       0: 67, 1: 19    |       ratio: 3.5263
Overall         |       0: 357, 1: 71   |       ratio: 5.0282
```

3.0.2 3.2.2 Use three algorithms from Problem 3.1 to train models on the training set. Please report the confusion matrix on the test set.

Code Clip 3.2.2

[23]:
```
# Code Clip 3.2.2
def eval_cm(model_class, X_train, y_train, X_test, y_test, folds, class_weight,␣
 ↪max_depth, **model_args):
    model = model_class(**model_args, class_weight=class_weight,␣
 ↪max_depth=max_depth)

    # train on entire training set and evaluate on test set
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    acc = accuracy_per_class(y_pred, y_test)

    return cm, acc


def compare_imbalanced(X_train, y_train, X_test, y_test, class_weight=None,␣
 ↪max_depth=None):
    # split training data into folds beforehand so that all methods use the␣
 ↪same folds for cross-validation
    kf = KFold(n_splits=5)
    folds = list(kf.split(X_train))

    cms = []
    accs = []

    #-------------Forest-------------
```

```
    cm, acc = eval_cm(RandomForestClassifier, X_train, y_train, X_test, y_test,␣
↪folds, class_weight, max_depth, n_estimators=50)
    cms.append(cm)
    accs.append(acc)

    #-------------CART-------------
    cm, acc = eval_cm(DecisionTreeClassifier, X_train, y_train, X_test, y_test,␣
↪folds, class_weight, max_depth, criterion='gini')
    cms.append(cm)
    accs.append(acc)

    #-------------ID3-------------
    cm, acc = eval_cm(DecisionTreeClassifier, X_train, y_train, X_test, y_test,␣
↪folds, class_weight, max_depth, criterion='entropy')
    cms.append(cm)
    accs.append(acc)

    return cms, accs


def print_cms(cms):
    print('Confusion Matrices\n' + '=' * 20 + '\n')
    names = ['Random Forest', 'CART', 'ID3']
    for name, cm in zip(names, cms):
        print(name)
        print('-' * 10)
        print(cm)
        print()
```

[24]:
```
cms, _ = compare_imbalanced(X_train, y_train, X_test, y_test)
print_cms(cms)
```

```
Confusion Matrices
====================

Random Forest
----------
[[66  1]
 [ 1 18]]

CART
----------
[[65  2]
 [ 3 16]]

ID3
----------
```

```
[[65  2]
 [ 3 16]]
```

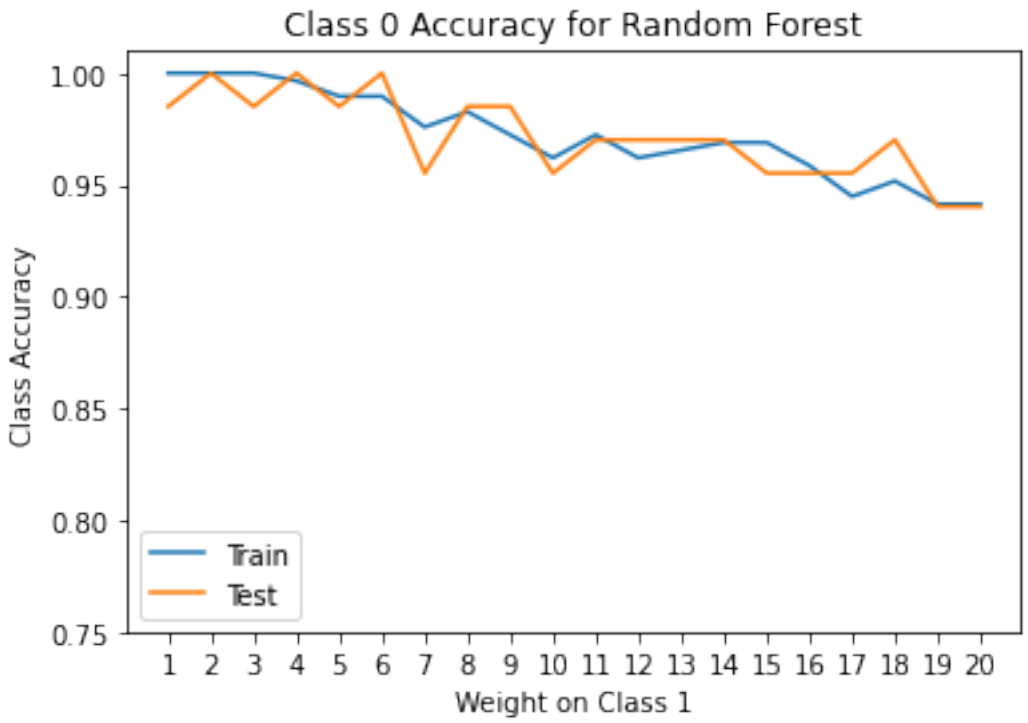
3.0.3 3.2.3 For each class, get the accuracy on the training set and testing set with different sample reweighting parameters. Plot them according to the reweight parameter. (Set the maximum depth of all the algorithms to 3).

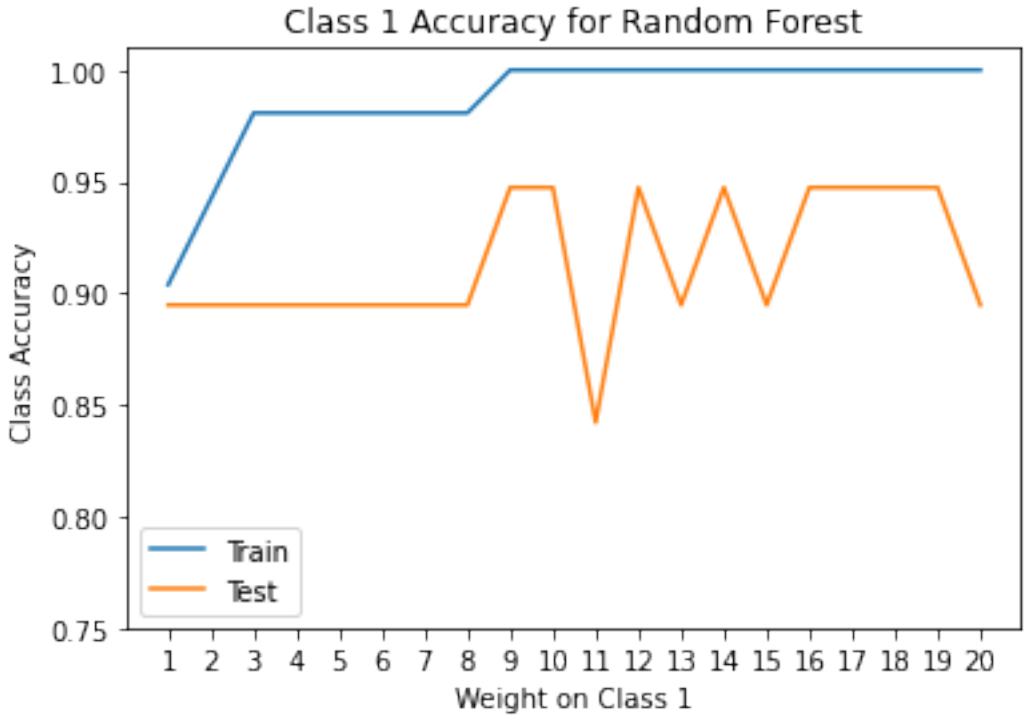
Code Clip 3.2.3:

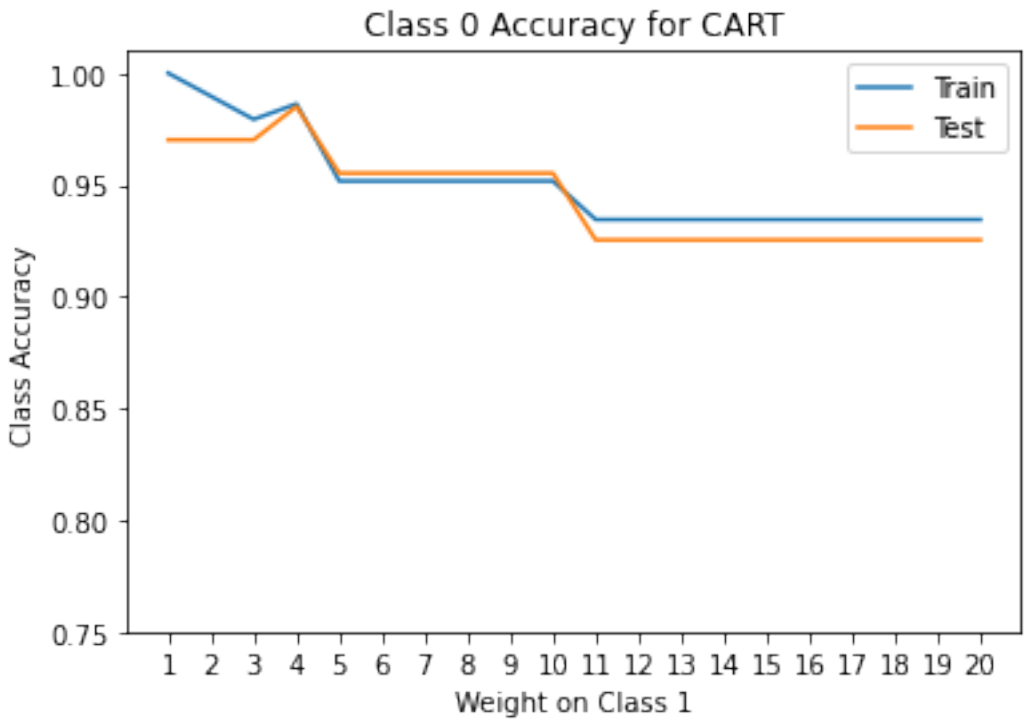
[25]:
```
# Code Clip 3.2.3
weight_list = list(np.arange(1, 21))

# storage for the class accuracies
train_accuracies = [
    {0: [], 1: []},    # index 0: random forest
    {0: [], 1: []},    # index 1: CART
    {0: [], 1: []}     # index 2: ID3
]

test_accuracies = [
    {0: [], 1: []},    # index 0: random forest
    {0: [], 1: []},    # index 1: CART
    {0: [], 1: []}     # index 2: ID3
]


def record_acc(model, idx):
    model.fit(X_train, y_train)
    acc = accuracy_per_class(model.predict(X_train), y_train)
    train_accuracies[idx][0].append(acc[0])
    train_accuracies[idx][1].append(acc[1])
    acc = accuracy_per_class(model.predict(X_test), y_test)
    test_accuracies[idx][0].append(acc[0])
    test_accuracies[idx][1].append(acc[1])


for weight in weight_list:
    # record training and test accuracies per class for each algorithm
    # Random Forest
    forest = RandomForestClassifier(n_estimators=50, class_weight={0: 1, 1:␣
 ↪weight}, max_depth=3)
    record_acc(forest, 0)

    # CART
    cart = DecisionTreeClassifier(criterion='gini', class_weight={0: 1, 1:␣
 ↪weight}, max_depth=3)
    record_acc(cart, 1)
```

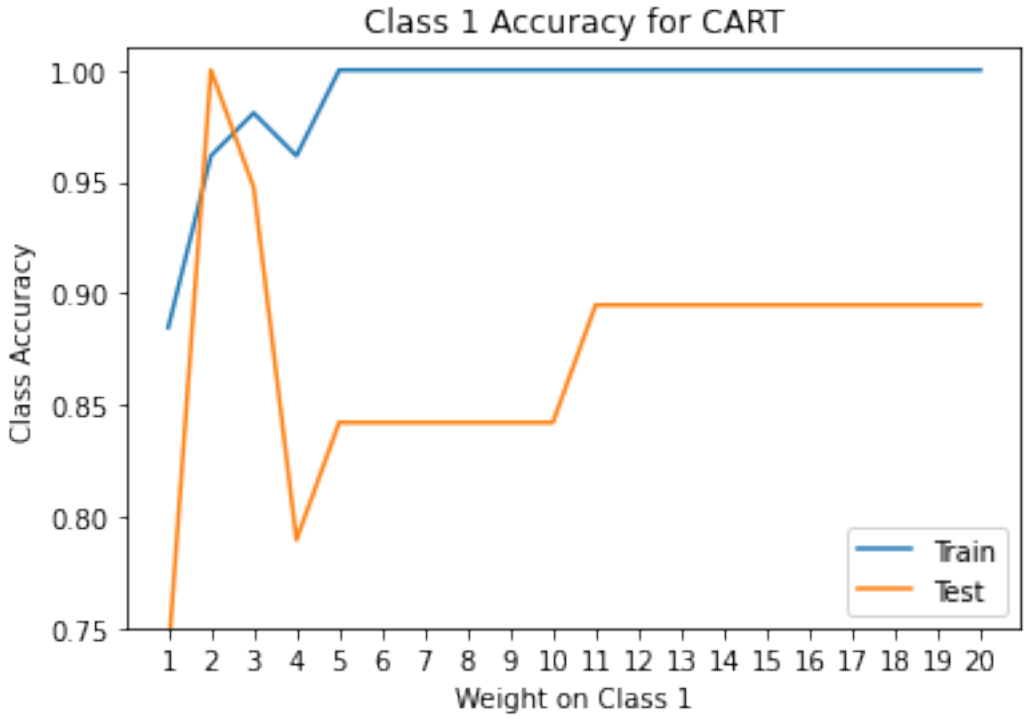
```
    # ID3
    id3 = RandomForestClassifier(criterion='entropy', class_weight={0: 1, 1:␣
↪weight}, max_depth=3)
    record_acc(id3, 2)
```

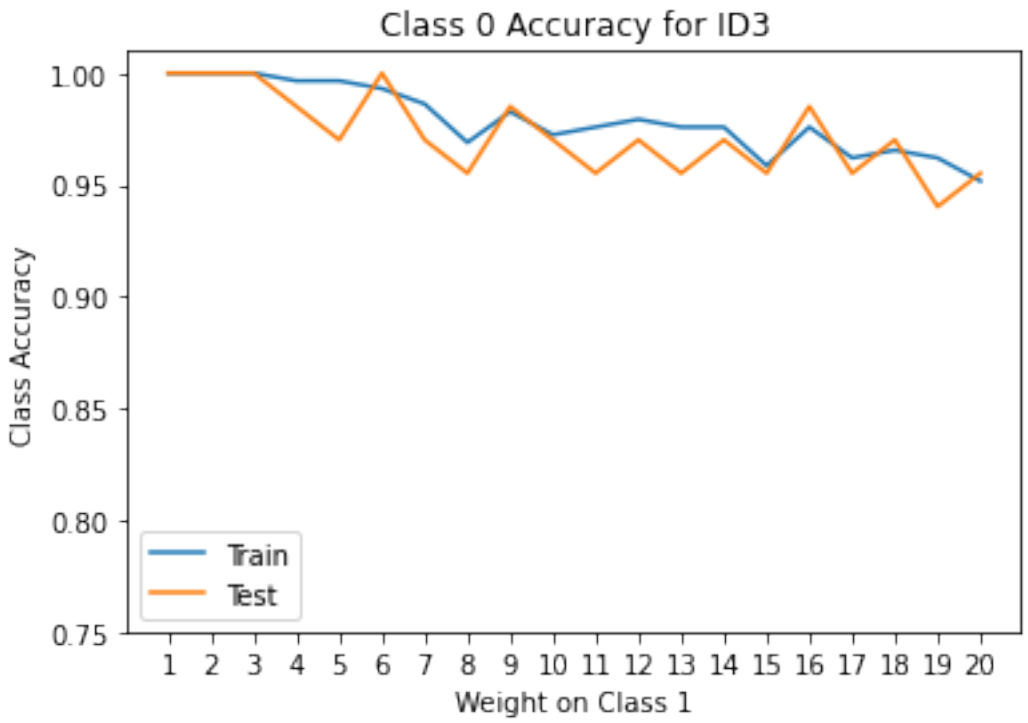
[26]:
```
# plot the accuracies vs the weight of class 1
def plot_accuracies(algo_idx, label):
    names = ['Random Forest', 'CART', 'ID3']
    plt.plot(train_accuracies[algo_idx][label], label='Train')
    plt.plot(test_accuracies[algo_idx][label], label='Test')
#     plt.plot([1] * len(weight_list), 'k--')
    plt.title(f'Class {label} Accuracy for {names[algo_idx]}')
    plt.xticks(ticks=range(len(weight_list)), labels=weight_list)
    plt.ylim(0.75, 1.01)
    plt.xlabel('Weight on Class 1')
    plt.ylabel('Class Accuracy')
    plt.legend()
    plt.show()


# Random Forest
plot_accuracies(0, 0)
plot_accuracies(0, 1)

# CART
plot_accuracies(1, 0)
plot_accuracies(1, 1)

# ID3
plot_accuracies(2, 0)
plot_accuracies(2, 1)
```

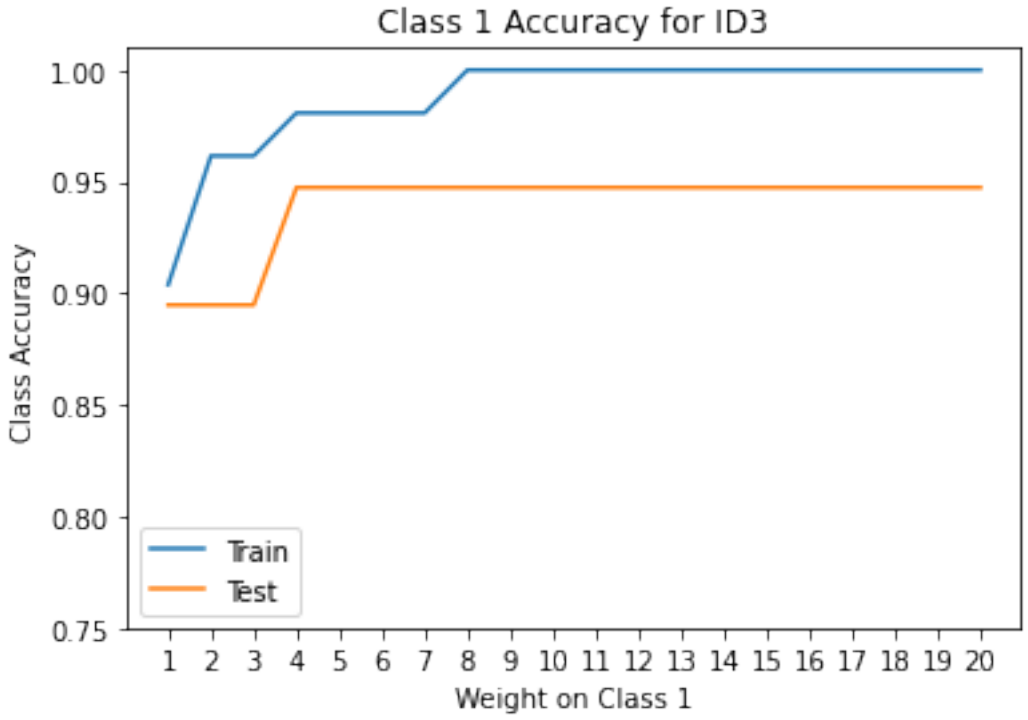

Class 0 Accuracy for Random Forest
Class Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
Train
Test
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Weight on Class 1


Class 1 Accuracy for Random Forest
Class Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
Train
Test
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Weight on Class 1


Class 0 Accuracy for CART
Train
Test
Class Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Weight on Class 1


Class 1 Accuracy for CART
Train
Test
Class Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Weight on Class 1


Class 0 Accuracy for ID3
Class Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
Train
Test
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Weight on Class 1


Class 1 Accuracy for ID3
Class Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
Train
Test
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Weight on Class 1