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.

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

In [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_ind
import timeit
```

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

In [3]:
```
train_data = pd.read_csv('./data_train.csv')
test_data = pd.read_csv('./data_test.csv')
```

In [4]:
```
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']
```

3.1 Balanced Dataset

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`.

In [5]:
```
from sklearn.metrics import confusion_matrix
def accuracy_all(predict, label):
    cm = confusion_matrix(label, predict)
    return np.sum(np.diag(cm))/np.sum(np.sum(cm))
```

In [6]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold

def compare_algorithms(X_train, y_train):
    score = pd.DataFrame()
    accuracy = pd.DataFrame()
    time_cost = pd.DataFrame()
    stats = pd.DataFrame()
    t_stats = pd.DataFrame()
    X_train_cross = pd.DataFrame()
    X_test_cross = pd.DataFrame()
    y_train_cross = pd.DataFrame()
    y_test_cross = pd.DataFrame()

    kf = KFold(n_splits=5)
    kf.get_n_splits(X_train)

    i = 0
    for train_index, test_index in kf.split(X_train):
        # print("TRAIN:", train_index, "TEST:", test_index)

        X_train_cross, X_test_cross = X_train.loc[train_index], X_train.loc[te
st_index]
        y_train_cross, y_test_cross = y_train.loc[train_index], y_train.loc[te
st_index]


        start = timeit.default_timer()
        clf_ID3 = DecisionTreeClassifier(criterion="entropy")
        clf_ID3 = clf_ID3.fit(X_train_cross,y_train_cross)
        score.at[i,'ID3'] = clf_ID3.score(X_test_cross,y_test_cross)
        y_predict = clf_ID3.predict(X_test_cross)
        accuracy.at[i,'ID3'] = accuracy_all(y_predict, y_test_cross)
        stop = timeit.default_timer()
        time_cost.at[i,'ID3'] = stop - start


        start = timeit.default_timer()
        clf_CART = DecisionTreeClassifier(criterion="gini")
        clf_CART = clf_CART.fit(X_train_cross,y_train_cross)
        score.at[i,'CART'] = clf_CART.score(X_test_cross,y_test_cross)
        y_predict = clf_CART.predict(X_test_cross)
        accuracy.at[i,'CART'] = accuracy_all(y_predict, y_test_cross)
        stop = timeit.default_timer()
        stop = timeit.default_timer()
        time_cost.at[i,'CART'] = stop - start



        start = timeit.default_timer()
        clf_RF =  RandomForestClassifier(n_estimators = 50, criterion="gini")
        clf_RF = clf_RF.fit(X_train_cross,y_train_cross)
        score.at[i,'RF'] = clf_RF.score(X_test_cross,y_test_cross)
        y_predict = clf_RF.predict(X_test_cross)
```

```
        accuracy.at[i,'RF'] = accuracy_all(y_predict, y_test_cross)
        stop = timeit.default_timer()
        stop = timeit.default_timer()
        time_cost.at[i,'RF'] = stop - start

        i = i+1

    stats["Means"] = accuracy.mean()
    stats["Standard Deviations"] = accuracy.std()

    a = accuracy['ID3'].to_numpy()
    b = accuracy['CART'].to_numpy()
    t_stats.at['ID3 & CART','ttest'], t_stats.at['ID3 & CART','p-value']= ttes
t_ind(a,b)

    b = accuracy['ID3'].to_numpy()
    a = accuracy['RF'].to_numpy()
    t_stats.at['ID3 & RF','ttest'], t_stats.at['ID3 & RF','p-value']=ttest_ind
(a,b)

    b = accuracy['CART'].to_numpy()
    a = accuracy['RF'].to_numpy()
    t_stats.at['CART & RF','ttest'], t_stats.at['CART & RF','p-value'] =ttest_
ind(a,b)

    return stats, t_stats, score, accuracy, time_cost
```

In [7]:
```
stats, t_stats, score, accuracy, time_cost = compare_algorithms(X_train, y_tra
in)
```

In [8]:
```
stats
```

Out[8]:

	Means	Standard Deviations
ID3	0.949451	0.022787
CART	0.929670	0.016666
RF	0.953846	0.016299

In [9]:
```
t_stats
```

Out[9]:

	ttest	p-value
ID3 & CART	1.566699	0.155819
ID3 & RF	0.350823	0.734780
CART & RF	2.319004	0.048996

The average accuracy for ID3 is 0.95, for CART is 0.93, and for Random Forest is 0.95. The standard deviations of accuracy for ID3 is 0.02, for CART is 0.02, and for Random Forest is 0.02. The t-test shows that, if the threshold is 0.1, the random forest algorithm is significantly better than CART algorithm; the different between random forest and ID3 is not singificant under this critiria.

In [10]:
```
score
```

Out[10]:

	ID3	CART	RF
0	0.967033	0.912088	0.978022
1	0.945055	0.934066	0.934066
2	0.934066	0.923077	0.956044
3	0.978022	0.956044	0.956044
4	0.923077	0.923077	0.945055

In [11]:
```
accuracy
```

Out[11]:

	ID3	CART	RF
0	0.967033	0.912088	0.978022
1	0.945055	0.934066	0.934066
2	0.934066	0.923077	0.956044
3	0.978022	0.956044	0.956044
4	0.923077	0.923077	0.945055

In [12]:
```
time_cost
```

Out[12]:

	ID3	CART	RF
0	0.010895	0.008172	0.083852
1	0.008787	0.005738	0.077919
2	0.007826	0.005241	0.082055
3	0.007494	0.007380	0.077848
4	0.007404	0.006787	0.081429

3.1.2 Effect of the depth.

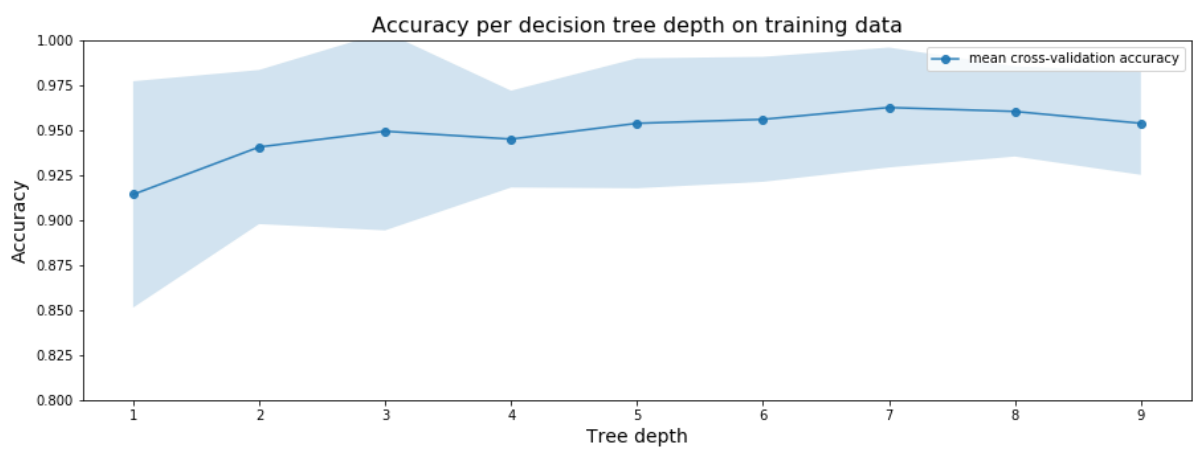
Code Clip 3.1.2: Complete the function `run_cross_validation_on_trees` .

In [13]:
```
def run_cross_validation_on_trees(X, y, tree_depths, cv=5, scoring='accuracy'
):
    cv_scores_list = []
    cv_scores_std = []
    cv_scores_mean = []
    accuracy_scores = []

    accuracy = pd.DataFrame()
    stats = pd.DataFrame()

    for depth in tree_depths:
        # Get the accuracy, mean, std from the cross validation.
        kf = KFold(n_splits=5)
        i = 0
        for train_index, test_index in kf.split(X_train):
        # print("TRAIN:", train_index, "TEST:", test_index)

            X_train_cross, X_test_cross = X_train.loc[train_index], X_train.lo
c[test_index]
            y_train_cross, y_test_cross = y_train.loc[train_index], y_train.lo
c[test_index]

            clf_RF =  RandomForestClassifier(max_depth = depth)
            clf_RF = clf_RF.fit(X_train_cross,y_train_cross)
            accuracy.at[i,depth] = clf_RF.score(X_test_cross,y_test_cross)
            i = i+1

    cv_scores_mean = accuracy.mean()
    cv_scores_std = accuracy.std()

    best_depth = cv_scores_mean.idxmax()

    clf_RF =  RandomForestClassifier(max_depth = depth)
    clf_RF = clf_RF.fit(X_train,y_train)
    accuracy_best = clf_RF.score(X_test,y_test)

    # print(best_depth,accuracy_best)

    cv_scores_mean = np.array(cv_scores_mean)
    cv_scores_std = np.array(cv_scores_std)
    # print(cv_scores_std)
    # print(cv_scores_mean)


    return cv_scores_mean, cv_scores_std, accuracy_scores, best_depth, accurac
y_best







# function for plotting cross-validation results
def plot_cross_validation_on_trees(depths, cv_scores_mean, cv_scores_std, accu
racy_scores, title):
    fig, ax = plt.subplots(1,1, figsize=(15,5))
```

```
    ax.plot(depths, cv_scores_mean, '-o', label='mean cross-validation accurac
y', alpha=0.9)
    ax.fill_between(depths, cv_scores_mean-2*cv_scores_std, cv_scores_mean+2*c
v_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()
```

In [14]:
```
sm_tree_depths = range(1,10)
sm_cv_scores_mean, sm_cv_scores_std, sm_accuracy_scores, best_depth, accuracy_
best = run_cross_validation_on_trees(X_train, y_train, sm_tree_depths)

print("The best depth: ", best_depth)
print("The accuracy with the best depth: ", accuracy_best)

# 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 d
ata')
plt.show()

# Play: Only uses the first ten features.
# sm_cv_scores_mean2, sm_cv_scores_std2, sm_accuracy_scores2 = run_cross_valid
ation_on_trees(X_train.iloc[:,:10],
#
y_train, sm_tree_depths)

# # plotting accuracy
# plot_cross_validation_on_trees(sm_tree_depths, sm_cv_scores_mean2, sm_cv_sco
res_std2, sm_accuracy_scores2,
#                                'Accuracy per decision tree depth on training
data')
# plt.show()
```

```
The best depth:  7
The accuracy with the best depth:  0.9649122807017544
```



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` .

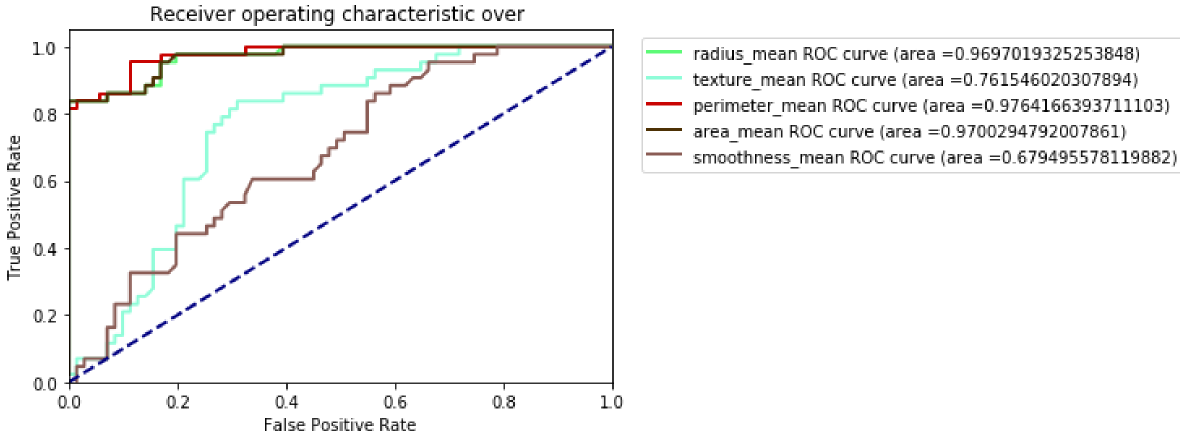
In [30]:
```
import random

def plot_roc(fpr, tpr, roc_auc, title = ''):
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area ={})'.format(roc_auc))
    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))
    plt.legend(loc="lower right")
    plt.show()

def plot_roc_multi(fpr, tpr, roc_auc, title = ''):
    plt.figure()
    lw = 2
    for k in range(6):
        for i in range(k*5,k*5+5):
            r = random.random()
            b = random.random()
            g = random.random()
            color_temp = (r, g, b)
            plt.plot(fpr[i], tpr[i], color=color_temp,
                 lw=lw, label='{} ROC curve (area ={})'.format(X_test.columns[
i],roc_auc.at[1,i]))
        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))
        plt.legend(bbox_to_anchor=(1.04,1), loc="upper left")
        plt.show()


    for i in range(30):
        r = random.random()
        b = random.random()
        g = random.random()
        color_temp = (r, g, b)
        plt.plot(fpr[i], tpr[i], color=color_temp,
             lw=lw, label='{} ROC curve (area ={})'.format(X_test.columns[i],r
oc_auc.at[1,i]))
    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))
    plt.legend(bbox_to_anchor=(1.04,1), loc="upper left")
    plt.show()
```

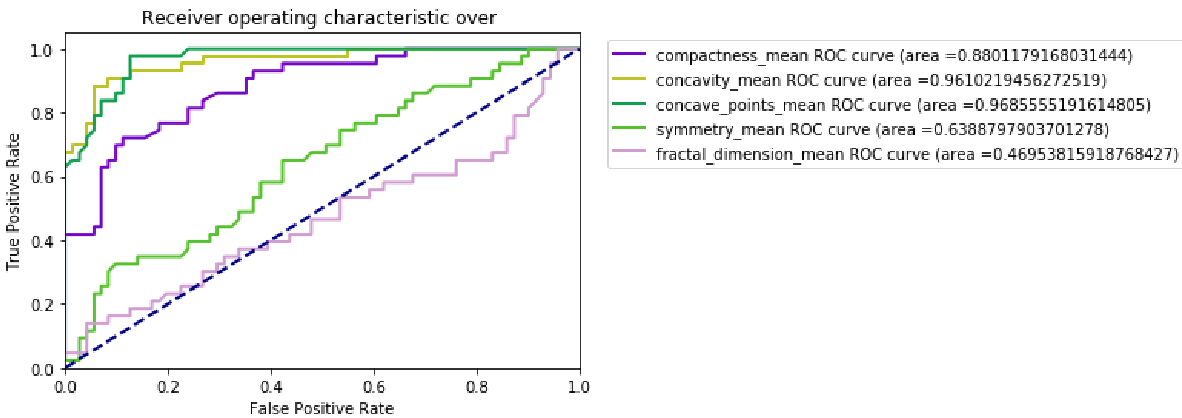
```
def draw_roc_with_feature_idx(X_test, y_test, i, draw = False):

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

    TPR = []
    FPR = []
    ROC = []
    AUC = []
    X_columns = list(X_test.columns)
    max_value = X_test[X_columns[i]].max()
    min_value = X_test[X_columns[i]].min()
    for j in np.arange(min_value-0.000001, max_value+0.000001, (max_value-min_
value)/1000):
        TPR_temp = 0
        FPR_temp = 0
        num_Pos = 0
        num_Neg = 0
        for k in range(0,len(X_test.index)):
            if X_test.at[k, X_columns[i]] > j and y_test[k] == 1:
                    TPR_temp = TPR_temp + 1
            if X_test.at[k, X_columns[i]] > j and y_test[k] == 0:
                    FPR_temp = FPR_temp + 1
        for k in range(0,len(X_test.index)):
            if y_test[k] == 1:
                num_Pos = num_Pos + 1
            if y_test[k] == 0:
                num_Neg = num_Neg + 1

        TPR_temp = TPR_temp / num_Pos
        FPR_temp = FPR_temp / num_Neg

        TPR.append(TPR_temp)
        FPR.append(FPR_temp)

    ROC_AUC = metrics.auc(FPR, TPR)

    if draw:
        plot_roc(FPR, TPR, ROC_AUC, title = 'Feature' + str(i))

    return ROC_AUC, TPR, FPR
```

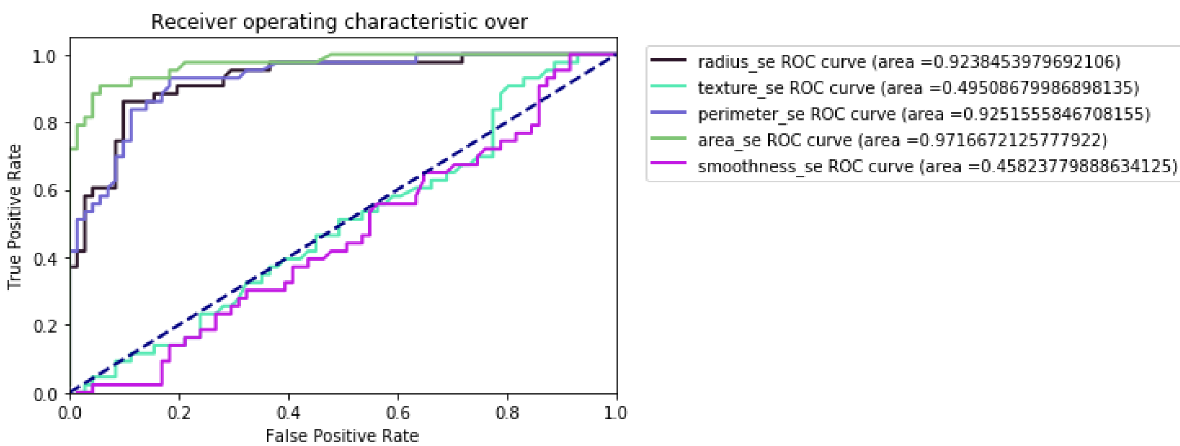
In [18]:
```
TPR_temp = pd.DataFrame()
FPR_temp = pd.DataFrame()
TPR_all = pd.DataFrame()
FPR_all = pd.DataFrame()
auc_all = pd.DataFrame()
for i in range(30):
    print(i, end = ' ')
    auc_temp, TPR_temp, FPR_temp = draw_roc_with_feature_idx(X_test, y_test, i
, draw = False)
    TPR_all.at[:,i] = TPR_temp
    FPR_all.at[:,i] = FPR_temp
    auc_all.at[1,i] = auc_temp
```

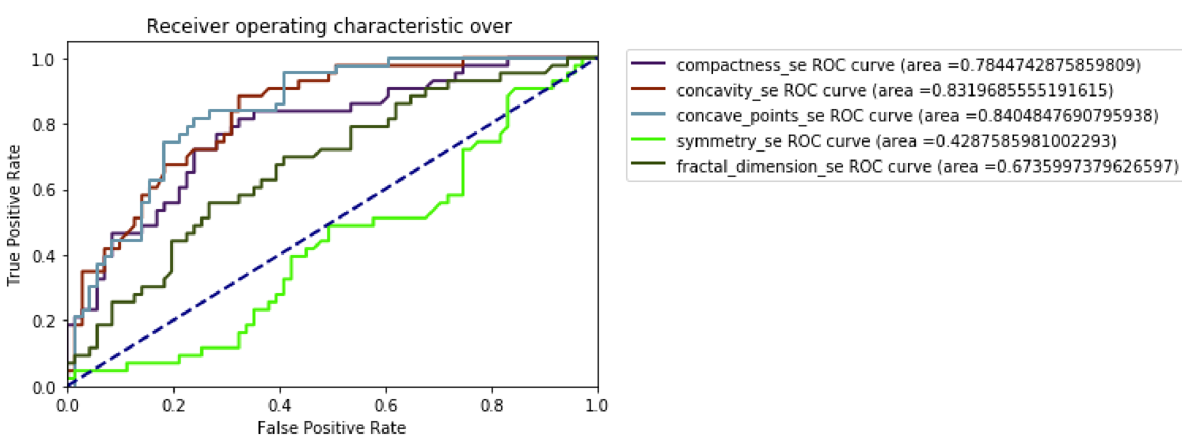
```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29
```

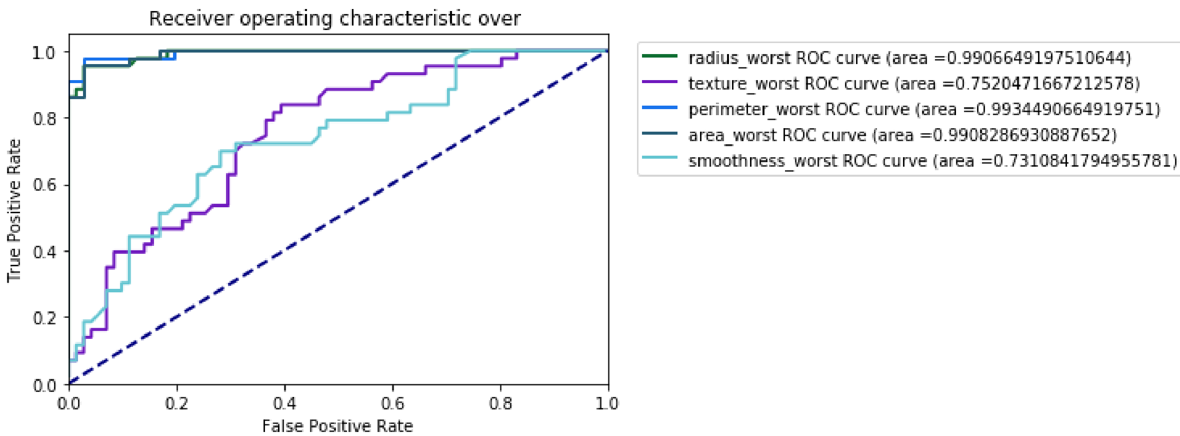
In [31]:
```
plot_roc_multi(FPR_all,TPR_all, auc_all, title = '')
```

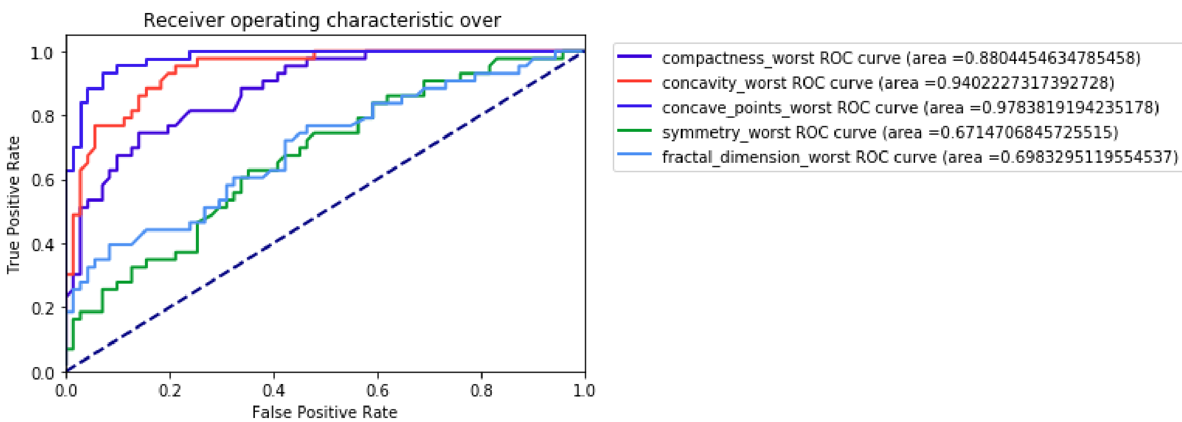

Receiver operating characteristic over
True Positive Rate
False Positive Rate
radius_mean ROC curve (area =0.9697019325253848)
texture_mean ROC curve (area =0.761546020307894)
perimeter_mean ROC curve (area =0.9764166393711103)
area_mean ROC curve (area =0.9700294792007861)
smoothness_mean ROC curve (area =0.679495578119882)

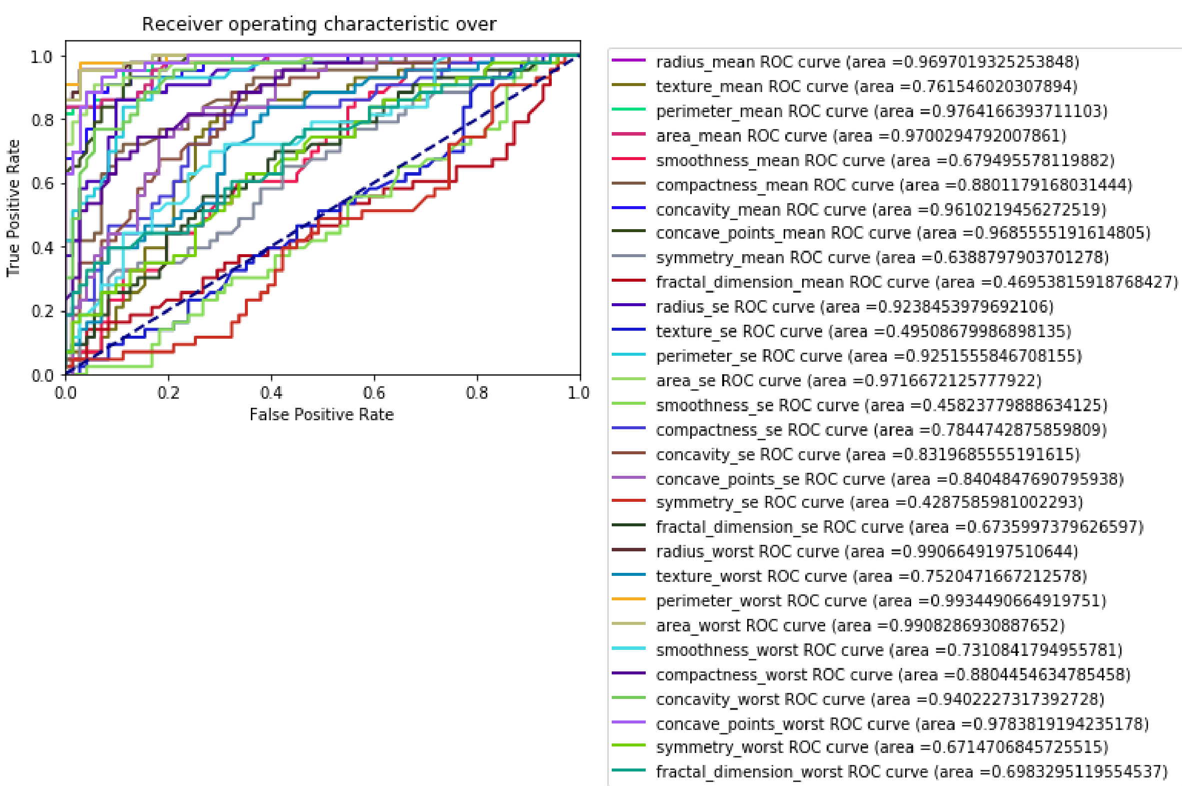

Receiver operating characteristic over
True Positive Rate
False Positive Rate
compactness_mean ROC curve (area =0.8801179168031444)
concavity_mean ROC curve (area =0.9610219456272519)
concave_points_mean ROC curve (area =0.9685555191614805)
symmetry_mean ROC curve (area =0.6388797903701278)
fractal_dimension_mean ROC curve (area =0.46953815918768427)


Receiver operating characteristic over
True Positive Rate
False Positive Rate
radius_se ROC curve (area =0.9238453979692106)
texture_se ROC curve (area =0.49508679986898135)
perimeter_se ROC curve (area =0.9251555846708155)
area_se ROC curve (area =0.9716672125777922)
smoothness_se ROC curve (area =0.45823779888634125)


Receiver operating characteristic over
True Positive Rate
False Positive Rate
compactness_se ROC curve (area =0.7844742875859809)
concavity_se ROC curve (area =0.8319685555191615)
concave_points_se ROC curve (area =0.8404847690795938)
symmetry_se ROC curve (area =0.4287585981002293)
fractal_dimension_se ROC curve (area =0.6735997379626597)







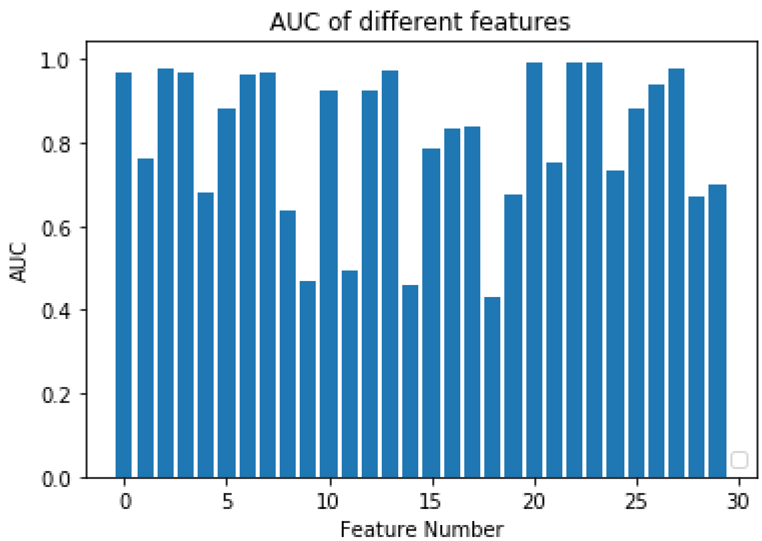
AUC of different features

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

In [28]:

```
plt.figure()

# Code Clip 3.1.3b

auc_draw = []
for i in range(30):
    auc_draw.append(auc_all.at[1,i])
plt.bar(range(X_test.shape[1]), auc_draw)
plt.xlabel('Feature Number')
plt.ylabel('AUC')
plt.title('AUC of different features')
plt.legend(loc="lower right")
plt.show()
```

```
No handles with labels found to put in legend.
```



In [29]:

```
# I report the important features with AUC > 0.8
for i in range(30):
    if auc_all.at[1,i] > 0.8:
        print(X_test.columns[i])
```

```
radius_mean
perimeter_mean
area_mean
compactness_mean
concavity_mean
concave_points_mean
radius_se
perimeter_se
area_se
concavity_se
concave_points_se
radius_worst
perimeter_worst
area_worst
compactness_worst
concavity_worst
concave_points_worst
```

In [33]:
```
# I also report the rank of importance of features (from important to less imp
rtant)

X_test.columns[np.argsort(auc_draw)[::-1]]
```

Out[33]:
```
Index(['perimeter_worst', 'area_worst', 'radius_worst', 'concave_points_wors
t',
       'perimeter_mean', 'area_se', 'area_mean', 'radius_mean',
       'concave_points_mean', 'concavity_mean', 'concavity_worst',
       'perimeter_se', 'radius_se', 'compactness_worst', 'compactness_mean',
       'concave_points_se', 'concavity_se', 'compactness_se', 'texture_mean',
       'texture_worst', 'smoothness_worst', 'fractal_dimension_worst',
       'smoothness_mean', 'fractal_dimension_se', 'symmetry_worst',
       'symmetry_mean', 'texture_se', 'fractal_dimension_mean',
       'smoothness_se', 'symmetry_se'],
      dtype='object')
```

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 $t_0 = 0, t_1 = 1$.

In [34]:
```
# Code Clip 3.1.4
t0 = 0
t1 = 0.2
TPR_work = []
NPR_work = []
for i in range(5):
    TPR_work  = TPR_all.iloc[:][i]
    FPR_work  = FPR_all.iloc[:][i]
    # delete all values out of the range(t0,t1)
    for j in range(0, len(TPR_work.index)):
        if FPR_work[j] > t1 or FPR_work[j] < t0:
            FPR_work[j] = -1
            TPR_work[j] = -1
    TPR_work = [x for x in TPR_work if x != -1]
    FPR_work = [x for x in FPR_work if x != -1]
    Partial_AUC = metrics.auc(FPR_work, TPR_work)
    Partial_AUC = Partial_AUC/(t1-t0)
    print(Partial_AUC)
```

```
0.8565345561742546
0.19079593842122503
0.8827382902063543
0.8589911562397641
0.20062233868326237
```

3.1.5 Model Reliance of CART model

Code Clip 3.1.5 Follow the instruction in the question.

In [35]:
```
feature = 'radius_mean'
train_data = pd.read_csv('./data_train.csv')
test_data = pd.read_csv('./data_test.csv')

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']

X_scramble = np.random.permutation(X_train[feature].values)
for i in range(0, len(X_scramble)):
    X_train.at[i,feature] = X_scramble[i]
```

In [39]:
```
# Code Clip 3.1.5


pd.options.display.max_columns = 100

train_data = pd.read_csv('./data_train.csv')
test_data = pd.read_csv('./data_test.csv')

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']

X = np.concatenate([X_train, X_test], axis= 0)
y = np.concatenate([y_train, y_test], axis= 0)

accuracy_reliance = pd.DataFrame()
clf_CART = DecisionTreeClassifier(criterion="gini")
clf_CART = clf_CART.fit(X_train,y_train)
accuracy_origin_test =  clf_CART.score(X_test,y_test)
accuracy_origin_entire = clf_CART.score(X,y)

for feature in features_mean:
    train_data = pd.read_csv('./data_train.csv')
    test_data = pd.read_csv('./data_test.csv')

    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']

    X_scramble = np.random.permutation(X_train[feature].values)
    for i in range(0, len(X_scramble)):
        X_train.at[i,feature] = X_scramble[i]

    clf_CART = DecisionTreeClassifier(criterion="gini")
    clf_CART = clf_CART.fit(X_train,y_train)
    accuracy_reliance.at['score_testset',feature] = clf_CART.score(X_test,y_te
st)
    # accuracy_reliance.at['loss_increase',feature] = accuracy_origin - accura
cy_reliance.at['score',feature]


    # I would like to calculate the score over the entire dataset as well
    train_data = pd.read_csv('./data_train.csv')
    test_data = pd.read_csv('./data_test.csv')

    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']

    X = np.concatenate([X_train, X_test], axis= 0)
```

```
    y = np.concatenate([y_train, y_test], axis= 0)

    accuracy_reliance.at['score_entireset',feature] = clf_CART.score(X,y)

    accuracy_reliance.at['loss_increase_testset',feature] = accuracy_origin_te
st - accuracy_reliance.at['score_testset',feature]
    accuracy_reliance.at['loss_increase_entireset',feature] = accuracy_origin_
entire - accuracy_reliance.at['score_entireset',feature]

accuracy_reliance
```

Out[39]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_me
score_testset	0.921053	0.938596	0.938596	0.938596	0.947:
score_entireset	0.984183	0.985940	0.985940	0.987698	0.989‹
loss_increase_testset	0.017544	0.000000	0.000000	0.000000	-0.0087
loss_increase_entireset	0.003515	0.001757	0.001757	0.000000	-0.0017

In [40]:
```
# for better output
accuracy_reliance_transposed = accuracy_reliance.T
accuracy_reliance_transposed
```

Out[40]:

	score_testset	score_entireset	loss_increase_testset	loss_increase_entire
radius_mean	0.921053	0.984183	0.017544	0.003
texture_mean	0.938596	0.985940	0.000000	0.001
perimeter_mean	0.938596	0.985940	0.000000	0.001
area_mean	0.938596	0.987698	0.000000	0.000
smoothness_mean	0.947368	0.989455	-0.008772	-0.001
compactness_mean	0.947368	0.968366	-0.008772	0.019
concavity_mean	0.938596	0.984183	0.000000	0.003
concave_points_mean	0.947368	0.978910	-0.008772	0.008
symmetry_mean	0.947368	0.989455	-0.008772	-0.001
fractal_dimension_mean	0.938596	0.987698	0.000000	0.000
radius_se	0.929825	0.985940	0.008772	0.001
texture_se	0.929825	0.985940	0.008772	0.001
perimeter_se	0.938596	0.987698	0.000000	0.000
area_se	0.929825	0.984183	0.008772	0.003
smoothness_se	0.938596	0.980668	0.000000	0.007
compactness_se	0.929825	0.985940	0.008772	0.001
concavity_se	0.921053	0.975395	0.017544	0.012
concave_points_se	0.929825	0.985940	0.008772	0.001
symmetry_se	0.938596	0.987698	0.000000	0.000
fractal_dimension_se	0.947368	0.989455	-0.008772	-0.001
radius_worst	0.938596	0.987698	0.000000	0.000
texture_worst	0.938596	0.987698	0.000000	0.000
perimeter_worst	0.929825	0.985940	0.008772	0.001
area_worst	0.929825	0.985940	0.008772	0.001
smoothness_worst	0.938596	0.987698	0.000000	0.000
compactness_worst	0.938596	0.987698	0.000000	0.000
concavity_worst	0.947368	0.989455	-0.008772	-0.001
concave_points_worst	0.964912	0.992970	-0.026316	-0.005
symmetry_worst	0.929825	0.985940	0.008772	0.001
fractal_dimension_worst	0.938596	0.987698	0.000000	0.000

3.2 Imbalanced Dataset

3.2.1： What is the ratio between the two labels ?

Code Clip 3.2.1

In [43]:
```
# Code Clip 3.2.1
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']

X = np.concatenate([X_train, X_test], axis= 0)
y = np.concatenate([y_train, y_test], axis= 0)

imbalance_ratio = pd.DataFrame()

pos = 0
neg = 0
for i in range(0,len(y_train)):
    if y_train[i] == 0:
        neg = neg + 1
    elif y_train[i] == 1:
        pos = pos + 1
imbalance_ratio.at[1,'training set'] = neg/pos

pos = 0
neg = 0
for i in range(0,len(y_test)):
    if y_test[i] == 0:
        neg = neg + 1
    elif y_test[i] == 1:
        pos = pos + 1
imbalance_ratio.at[1,'test set'] = neg/pos

pos = 0
neg = 0
for i in range(0,len(y)):
    if y[i] == 0:
        neg = neg + 1
    elif y[i] == 1:
        pos = pos + 1
imbalance_ratio.at[1,'entire set'] = neg/pos
```

In [44]:
```
imbalance_ratio
```

Out[44]:

	training set	test set	entire set
1	5.576923	3.526316	5.028169

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

In [45]:
```
# Code Clip 3.2.2
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)

clf_ID3 = DecisionTreeClassifier(criterion="entropy")
clf_ID3 = clf_ID3.fit(X_train,y_train)
y_predict = clf_ID3.predict(X_test)
cm_ID3 = confusion_matrix(y_test, y_predict)


clf_CART = DecisionTreeClassifier(criterion="gini")
clf_CART = clf_CART.fit(X_train,y_train)
y_predict = clf_CART.predict(X_test)
cm_CART = confusion_matrix(y_test, y_predict)

clf_RF =  RandomForestClassifier(n_estimators = 50, criterion="gini")
clf_RF = clf_RF.fit(X_train,y_train)
y_predict = clf_RF.predict(X_test)
cm_RF = confusion_matrix(y_test, y_predict)
```

In [46]:
```
cm_ID3
```

Out[46]:
```
array([[63,  4],
       [ 3, 16]], dtype=int64)
```

In [47]:
```
cm_CART
```

Out[47]:
```
array([[65,  2],
       [ 2, 17]], dtype=int64)
```

In [48]:
```
cm_RF
```

Out[48]:
```
array([[66,  1],
       [ 1, 18]], dtype=int64)
```

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:

In [50]:
```
from sklearn import tree
def accuracy_per_class(predict, label):
    cm = confusion_matrix(label, predict)
    return np.diag(cm)/np.sum(cm, axis = 1)

weight_list = list(np.arange(1, 21))

accuracy_ID3 =pd.DataFrame()
accuracy_CART =pd.DataFrame()
accuracy_RF =pd.DataFrame()

i = 0
for weight in weight_list:
    # Code Clip 3.2.3
    # Calculte the accuracy of each class.

    weight_temp = {0: 1, 1: weight}

    clf_ID3 = DecisionTreeClassifier(criterion="entropy",max_depth = 3,class_w
eight = weight_temp)
    clf_ID3 = clf_ID3.fit(X_train,y_train)
    y_predict = clf_ID3.predict(X_test)
    accuracy_temp = accuracy_per_class(y_predict, y_test)
    accuracy_ID3.at[i, 'weight'],accuracy_ID3.at[i, 'negative_testing'], accur
acy_ID3.at[i, 'positive_testing'] = weight,accuracy_temp[0],accuracy_temp[1]
    y_predict = clf_ID3.predict(X_train)
    accuracy_temp = accuracy_per_class(y_predict, y_train)
    accuracy_ID3.at[i, 'negative_training'], accuracy_ID3.at[i, 'positive_trai
ning'] = accuracy_temp[0],accuracy_temp[1]

    clf_CART = DecisionTreeClassifier(criterion="gini",max_depth = 3,class_wei
ght = weight_temp)
    clf_CART = clf_CART.fit(X_train,y_train)
    y_predict = clf_CART.predict(X_test)
    accuracy_temp = accuracy_per_class(y_predict, y_test)
    accuracy_CART.at[i, 'weight'],accuracy_CART.at[i, 'negative_testing'], acc
uracy_CART.at[i, 'positive_testing'] = weight,accuracy_temp[0],accuracy_temp[1
]
    y_predict = clf_CART.predict(X_train)
    accuracy_temp = accuracy_per_class(y_predict, y_train)
    accuracy_CART.at[i, 'negative_training'], accuracy_CART.at[i, 'positive_tr
aining'] = accuracy_temp[0],accuracy_temp[1]


    clf_RF =  RandomForestClassifier(n_estimators = 50, criterion="gini",max_d
epth = 3,class_weight = weight_temp)
    clf_RF = clf_RF.fit(X_train,y_train)
    y_predict = clf_RF.predict(X_test)
    accuracy_temp = accuracy_per_class(y_predict, y_test)
    accuracy_RF.at[i, 'weight'],accuracy_RF.at[i, 'negative_testing'], accurac
y_RF.at[i, 'positive_testing'] = weight,accuracy_temp[0],accuracy_temp[1]
    y_predict = clf_RF.predict(X_train)
    accuracy_temp = accuracy_per_class(y_predict, y_train)
    accuracy_RF.at[i, 'negative_training'], accuracy_RF.at[i, 'positive_traini
ng'] = accuracy_temp[0],accuracy_temp[1]
```

```
    i = i+1
```

In [52]:
```
accuracy_ID3
```

Out[52]:

	weight	negative_testing	positive_testing	negative_training	positive_training
0	1.0	0.985075	0.789474	0.996552	0.980769
1	2.0	0.970149	0.894737	1.000000	0.923077
2	3.0	1.000000	0.894737	0.996552	0.942308
3	4.0	0.985075	0.894737	0.996552	0.942308
4	5.0	0.985075	0.894737	0.996552	0.942308
5	6.0	0.955224	0.842105	0.944828	1.000000
6	7.0	0.955224	0.842105	0.944828	1.000000
7	8.0	0.955224	0.842105	0.944828	1.000000
8	9.0	0.955224	0.842105	0.944828	1.000000
9	10.0	0.955224	0.842105	0.944828	1.000000
10	11.0	0.955224	0.842105	0.944828	1.000000
11	12.0	0.955224	0.842105	0.944828	1.000000
12	13.0	0.955224	0.842105	0.944828	1.000000
13	14.0	0.955224	0.842105	0.944828	1.000000
14	15.0	0.955224	0.842105	0.944828	1.000000
15	16.0	0.955224	0.842105	0.944828	1.000000
16	17.0	0.955224	0.842105	0.944828	1.000000
17	18.0	0.955224	0.842105	0.944828	1.000000
18	19.0	0.955224	0.842105	0.944828	1.000000
19	20.0	0.955224	0.842105	0.944828	1.000000

In [53]:
```
accuracy_CART
```

Out[53]:

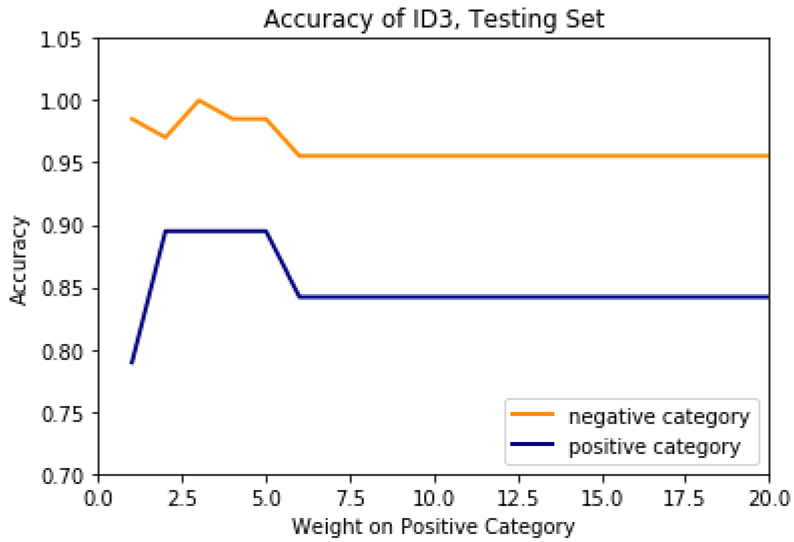
	weight	negative_testing	positive_testing	negative_training	positive_training
0	1.0	0.970149	0.842105	1.000000	0.884615
1	2.0	0.970149	1.000000	0.989655	0.961538
2	3.0	0.955224	0.947368	0.979310	0.980769
3	4.0	0.985075	0.789474	0.986207	0.961538
4	5.0	0.955224	0.842105	0.951724	1.000000
5	6.0	0.955224	0.842105	0.951724	1.000000
6	7.0	0.955224	0.842105	0.951724	1.000000
7	8.0	0.955224	0.842105	0.951724	1.000000
8	9.0	0.955224	0.842105	0.951724	1.000000
9	10.0	0.955224	0.842105	0.951724	1.000000
10	11.0	0.925373	0.894737	0.934483	1.000000
11	12.0	0.925373	0.894737	0.934483	1.000000
12	13.0	0.925373	0.894737	0.934483	1.000000
13	14.0	0.925373	0.894737	0.934483	1.000000
14	15.0	0.925373	0.894737	0.934483	1.000000
15	16.0	0.925373	0.894737	0.934483	1.000000
16	17.0	0.925373	0.894737	0.934483	1.000000
17	18.0	0.925373	0.894737	0.934483	1.000000
18	19.0	0.925373	0.894737	0.934483	1.000000
19	20.0	0.925373	0.894737	0.934483	1.000000

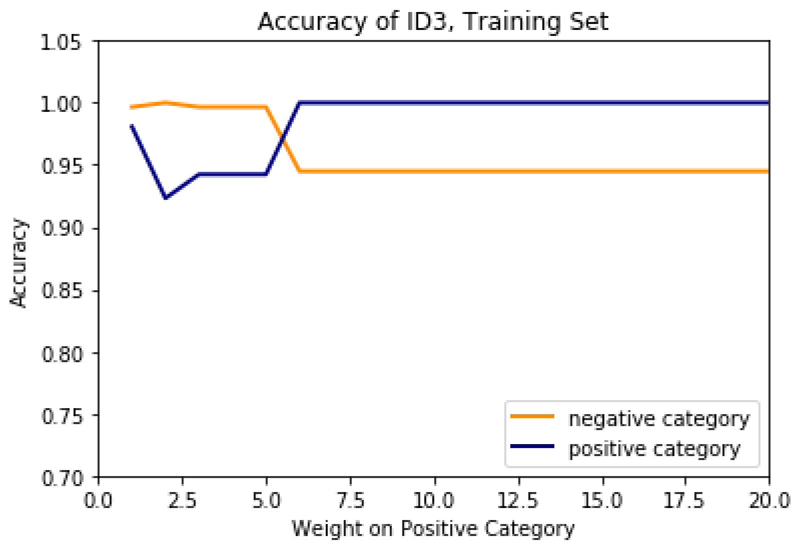
In [54]:
```
accuracy_RF
```

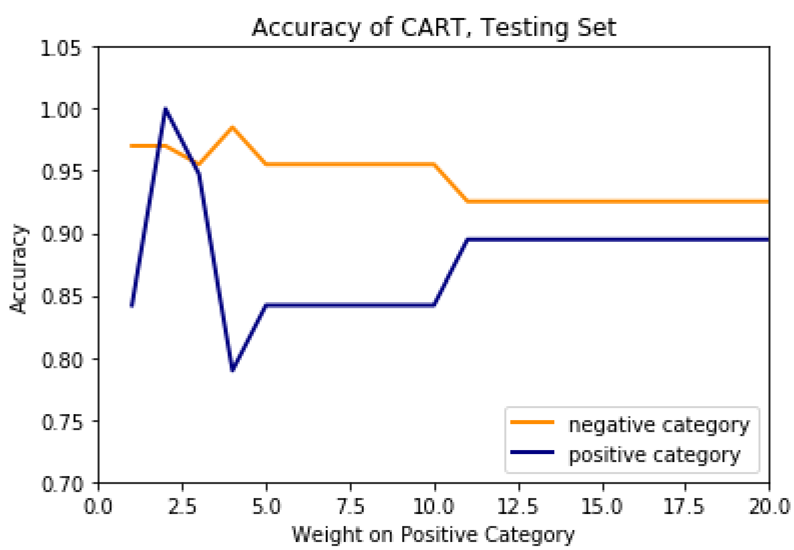
Out[54]:

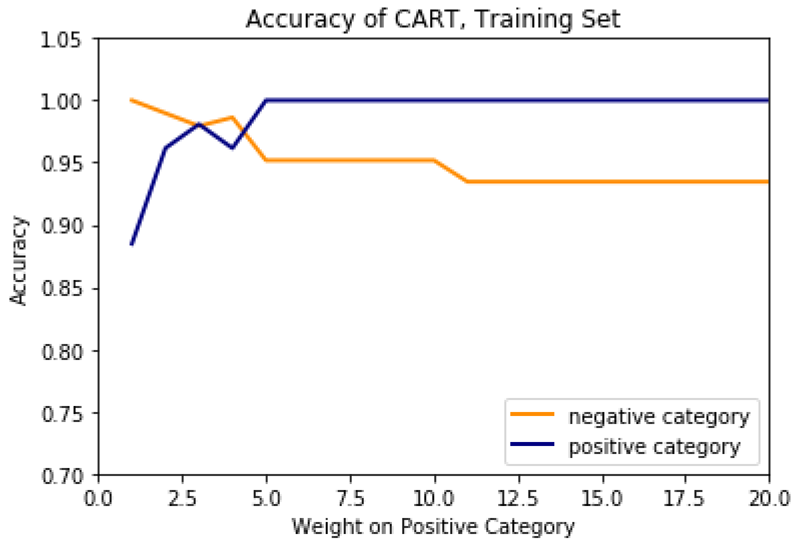
	weight	negative_testing	positive_testing	negative_training	positive_training
0	1.0	0.985075	0.842105	1.000000	0.903846
1	2.0	0.985075	0.947368	1.000000	0.961538
2	3.0	0.985075	0.947368	1.000000	0.961538
3	4.0	1.000000	0.894737	0.996552	0.980769
4	5.0	0.985075	0.894737	0.996552	0.980769
5	6.0	0.970149	0.894737	0.986207	1.000000
6	7.0	0.970149	0.947368	0.989655	1.000000
7	8.0	0.955224	0.894737	0.986207	1.000000
8	9.0	0.985075	0.947368	0.965517	1.000000
9	10.0	0.955224	0.894737	0.986207	1.000000
10	11.0	0.955224	0.947368	0.972414	0.980769
11	12.0	0.955224	0.947368	0.968966	1.000000
12	13.0	0.940299	0.947368	0.962069	1.000000
13	14.0	0.970149	0.947368	0.948276	1.000000
14	15.0	0.940299	0.947368	0.951724	1.000000
15	16.0	0.970149	0.947368	0.948276	0.980769
16	17.0	0.955224	0.894737	0.944828	1.000000
17	18.0	0.955224	0.894737	0.937931	1.000000
18	19.0	0.955224	0.947368	0.958621	0.980769
19	20.0	0.955224	0.947368	0.934483	1.000000

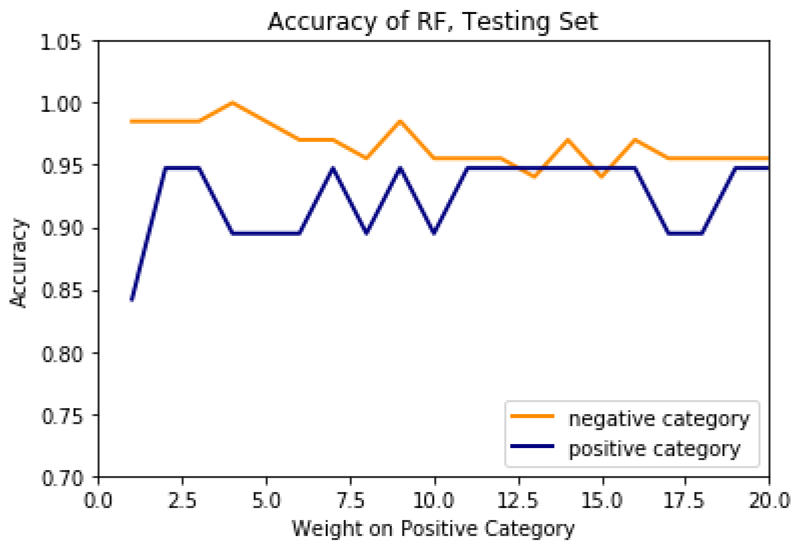
In [55]:
```
def plot_accuracy(x, y, z, title = ''):
    plt.figure()
    lw = 2
    plt.plot(x, y, color='darkorange',
             lw=lw, label='negative category')
    plt.plot(x, z, color='navy',
             lw=lw, label='positive category')
    plt.xlim([0.0, 20])
    plt.ylim([0.7, 1.05])
    plt.xlabel('Weight on Positive Category')
    plt.ylabel('Accuracy')
    plt.title('Accuracy of {}'.format(title))
    plt.legend(loc="lower right")
    plt.show()

x = accuracy_ID3['weight']
y = accuracy_ID3['negative_testing']
z = accuracy_ID3['positive_testing']
plot_accuracy(x, y, z, title = 'ID3, Testing Set')

x = accuracy_ID3['weight']
y = accuracy_ID3['negative_training']
z = accuracy_ID3['positive_training']
plot_accuracy(x, y, z, title = 'ID3, Training Set')

x = accuracy_CART['weight']
y = accuracy_CART['negative_testing']
z = accuracy_CART['positive_testing']
plot_accuracy(x, y, z, title = 'CART, Testing Set')

x = accuracy_CART['weight']
y = accuracy_CART['negative_training']
z = accuracy_CART['positive_training']
plot_accuracy(x, y, z, title = 'CART, Training Set')

x = accuracy_RF['weight']
y = accuracy_RF['negative_testing']
z = accuracy_RF['positive_testing']
plot_accuracy(x, y, z, title = 'RF, Testing Set')

x = accuracy_RF['weight']
y = accuracy_RF['negative_training']
z = accuracy_RF['positive_training']
plot_accuracy(x, y, z, title = 'RF, Training Set')
```

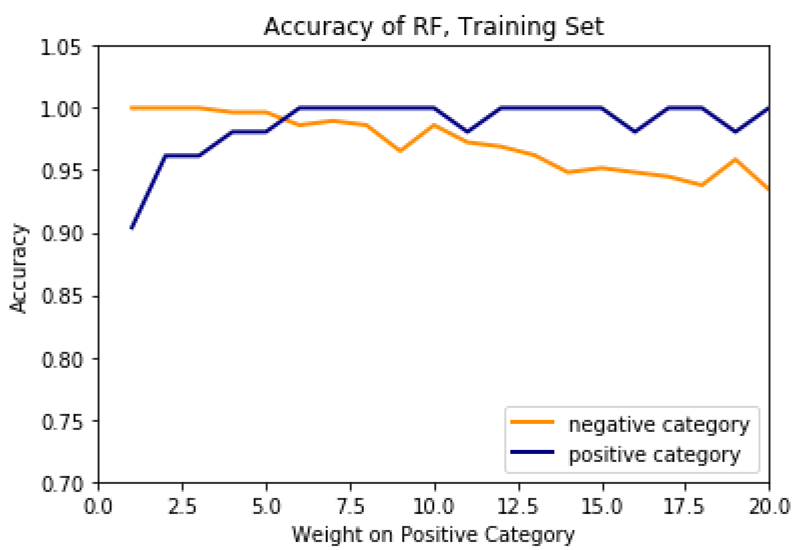

Accuracy of ID3, Testing Set
Accuracy
Weight on Positive Category
negative category
positive category


Accuracy of ID3, Training Set
Accuracy
Weight on Positive Category
negative category
positive category


Accuracy of CART, Testing Set
Accuracy
Weight on Positive Category
negative category
positive category


Accuracy of CART, Training Set
Accuracy
1.05
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Weight on Positive Category
negative category
positive category


Accuracy of RF, Testing Set
Accuracy
1.05
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Weight on Positive Category
negative category
positive category


Accuracy of RF, Training Set
Accuracy
1.05
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Weight on Positive Category
negative category
positive category

In [ ]:

```

```