In [49]:

```python
# Python ≥3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn ≥0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "ensembles"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)

options(jupyter.plot_mimetypes = c("text/plain", "image/png" ))
```

Prepare the data. Load breast cancer data

In [50]:

```python
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print(cancer.target[0:20])
print(list(cancer.target_names))
print(cancer.data[0:5])
print(list(cancer.feature_names))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
['malignant', 'benign']
[[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
  1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
  6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
  1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
  4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02
  7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.408e+01
  5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.499e+01
  2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01 1.860e-01
  2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01 1.974e-01
  1.279e-01 2.069e-01 5.999e-02 7.456e-01 7.869e-01 4.585e+00 9.403e+01
  6.150e-03 4.006e-02 3.832e-02 2.058e-02 2.250e-02 4.571e-03 2.357e+01
  2.553e+01 1.525e+02 1.709e+03 1.444e-01 4.245e-01 4.504e-01 2.430e-01
  3.613e-01 8.758e-02]
 [1.142e+01 2.038e+01 7.758e+01 3.861e+02 1.425e-01 2.839e-01 2.414e-01
  1.052e-01 2.597e-01 9.744e-02 4.956e-01 1.156e+00 3.445e+00 2.723e+01
  9.110e-03 7.458e-02 5.661e-02 1.867e-02 5.963e-02 9.208e-03 1.491e+01
  2.650e+01 9.887e+01 5.677e+02 2.098e-01 8.663e-01 6.869e-01 2.575e-01
  6.638e-01 1.730e-01]
 [2.029e+01 1.434e+01 1.351e+02 1.297e+03 1.003e-01 1.328e-01 1.980e-01
  1.043e-01 1.809e-01 5.883e-02 7.572e-01 7.813e-01 5.438e+00 9.444e+01
  1.149e-02 2.461e-02 5.688e-02 1.885e-02 1.756e-02 5.115e-03 2.254e+01
  1.667e+01 1.522e+02 1.575e+03 1.374e-01 2.050e-01 4.000e-01 1.625e-01
  2.364e-01 7.678e-02]]
['mean radius', 'mean texture', 'mean perimeter', 'mean area', 'mean smoothness', 'mean compactness', 'mean concavity
', 'mean concave points', 'mean symmetry', 'mean fractal dimension', 'radius error', 'texture error', 'perimeter erro
r', 'area error', 'smoothness error', 'compactness error', 'concavity error', 'concave points error', 'symmetry error
', 'fractal dimension error', 'worst radius', 'worst texture', 'worst perimeter', 'worst area', 'worst smoothness', '
worst compactness', 'worst concavity', 'worst concave points', 'worst symmetry', 'worst fractal dimension']
```

Problem 1.

Break the data into training (80%)/testing data(20%). Estimate a tree classification model with maximum depth of 2. Plot the tree and calculate the accuracy rate. Predict target using all features, don't forget to set random numbers to 42.

In [58]:

```python
# Starting point
import random
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import pydotplus
X = cancer.data
y = cancer.target
random.seed(42)
import os
```

In [59]:

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X_train, y_train)
```

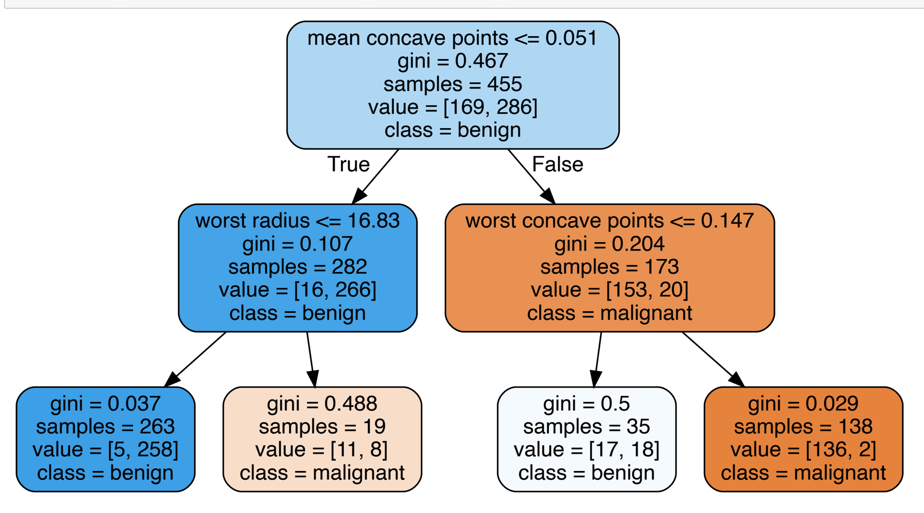
Out[59]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=2, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=42, splitter='best')
```

In [60]:

```python
from graphviz import Source
from sklearn.tree import export_graphviz

export_graphviz(
        tree_clf,
        out_file=os.path.join(IMAGES_PATH, "cancer_tree.dot"),
        feature_names=cancer.feature_names,
        class_names=cancer.target_names,
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "cancer_tree.dot"))
```

Out[60]:



In [60]:

In [61]:

```python
from sklearn.metrics import accuracy_score
y_pred = tree_clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Testing accuracy score for a tree with max depth 2 is", acc)
```

```
Testing accuracy score for a tree with max depth 2 is 0.9298245614035088
```

Problem 2: Estimate an unrestricted tree on training data and test it on testing data. Find two most important features and create a scatter plot of malignant and benign tumors along the two axes of two most important feature. Hint: For example of a graph look at: https://stackoverflow.com/questions/12487060/matplotlib-color-according-to-class-labels Do you think the data need rotation?

In [62]:

```python
tree_full = DecisionTreeClassifier(random_state=42)
tree_full.fit(X_train, y_train)
sorted_feat = tree_full.feature_importances_
for name, score in zip(cancer["feature_names"], tree_full.feature_importances_):
    print(name, score)
```

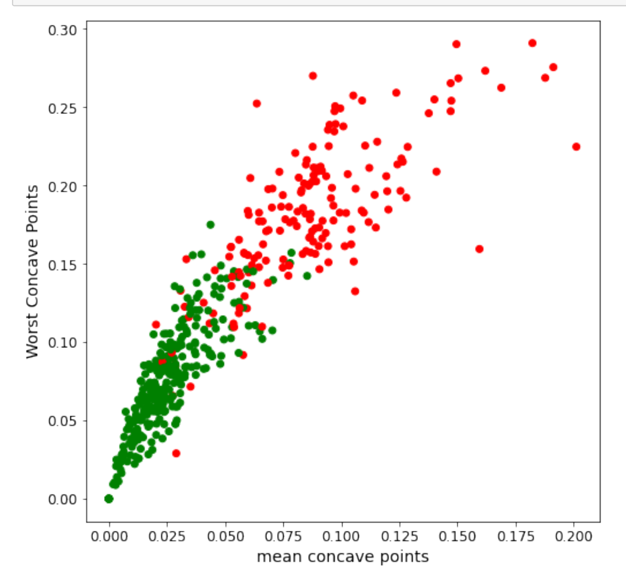
```
mean radius 0.0
mean texture 0.05847766231107586
mean perimeter 0.0
mean area 0.0
mean smoothness 0.0
mean compactness 0.0
mean concavity 0.0
mean concave points 0.6914195549049809
mean symmetry 0.0
mean fractal dimension 0.0
radius error 0.0
texture error 0.0
perimeter error 0.0
area error 0.011982573676838769
smoothness error 0.0012367800829339453
compactness error 0.0
concavity error 0.0062757755065447375
concave points error 0.015930814747382796
symmetry error 0.0
fractal dimension error 0.018554466715001834
worst radius 0.05229926933685694
worst texture 0.017445161675930944
worst perimeter 0.051493960584869665
worst area 0.0
worst smoothness 0.009233190446208121
worst compactness 0.0
worst concavity 0.0
worst concave points 0.06565079001137543
worst symmetry 0.0
worst fractal dimension 0.0
```

In [70]:

```python
worst_rad = X_train[:,7]
worst_conc_per = X_train[:,27]
malignancy = y_train
```

In [72]:

```python
from matplotlib.colors import ListedColormap, LinearSegmentedColormap
cmap = ListedColormap(['red','green'])
fig = plt.figure(figsize=(8,8))
plt.scatter(worst_rad, worst_conc_per, c=y_train, cmap=cmap)
plt.xlabel("mean concave points")
plt.ylabel("Worst Concave Points")
plt.show()
```



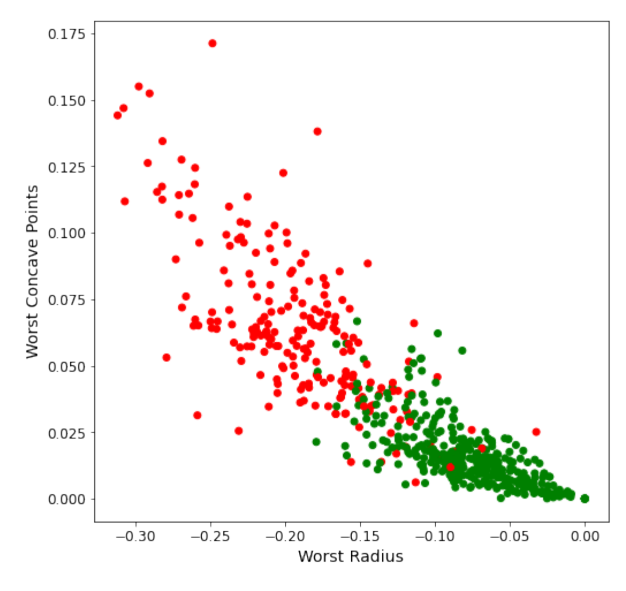
Problem 3

Report accuracy using top 2 features on the full data from problem 2. Loop over 10000 random number between -1 and 1 (from -3 to 3 radians) to find an optimal rotation angle. Report accuracy improvelemt over unrotated data.

In [77]:

```python
np.random.seed(42)

# Unrotated accuracy
Imp_tr = np.column_stack(( X_train[:,7],X_train[:,27]))
Imp_test = np.column_stack(( X_test[:,7],X_test[:,27]))
tree_clf.fit(Imp_tr, y_train)
y_pred = tree_clf.predict(Imp_test)
acc_best = accuracy_score(y_test, y_pred)
print(f'Unrotated accuracy is {acc_best}')

angle_list = np.random.uniform(-3,3,10000)

#Imp = np.column_stack((worst_rad, worst_conc_per))


for a in range(10000):
  angle = angle_list[a]
  rotation_matrix = np.array([[np.cos(angle), -np.sin(angle)], [np.sin(angle), np.cos(angle)]])
  Xr = Imp_tr.dot(rotation_matrix)
  Xr_test = Imp_test.dot(rotation_matrix)
  tree_clf.fit(Xr, y_train)
  y_pred = tree_clf.predict(Xr_test)
  acc_new = accuracy_score(y_test, y_pred)
  if acc_new > acc_best:
    acc_best = acc_new
    angle_best = angle


print(f'best rotated accuracy is {acc} , best angle is {angle_best}')

# Create rotaton matrix 2x2 for each point on a 2-dimensional plane
rotation_matrix = np.array([[np.cos(angle), -np.sin(angle)], [np.sin(angle), np.cos(angle)]])
Xr = Imp.dot(rotation_matrix)
colors = ['red','green']
fig = plt.figure(figsize=(8,8))
plt.scatter(Xr[:,0], Xr[:,1], c=y, cmap=ListedColormap(colors))
plt.xlabel("Worst Radius")
plt.ylabel("Worst Concave Points")
plt.show()
```

```
Unrotated accuracy is 0.9122807017543859
best rotated accuracy is 0.9298245614035088 , best angle is -1.0919791501688167
```



Problem 4

In the main data drop variables used in the problem 3. Add instead the rotated variables (substitution). Estimate accuracy score using with a tree classifier with max depth = 2 (Same as in problem 1). How much did we gain from rotation?

In [81]:

```python
rotation_matrix = np.array([[np.cos(angle_best), -np.sin(angle_best)], [np.sin(angle_best), np.cos(angle_best)]])
Xr = Imp_tr.dot(rotation_matrix)
Xr_test = Imp_test.dot(rotation_matrix)
X_new_tr  = X_train
X_new_test = X_test


X_new_tr[:,7] = Xr[:,0]
X_new_tr[:,27] = Xr[:,1]
X_new_test[:,7] = Xr_test[:,0]
X_new_test[:,27] = Xr_test[:,1]
```

In [84]:

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X_train, y_train)
y_pred = tree_clf.predict(X_test)
acc_not_rot = accuracy_score(y_test, y_pred)


X_new_tr, X_new_test, y_train, y_test = train_test_split(X_new, y, test_size=0.2, random_state=42)
tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X_new_tr, y_train)
y_pred = tree_clf.predict(X_new_test)
acc_r = accuracy_score(y_test, y_pred)
print("Testing accuracy score for a tree with max depth 2 is", acc_r)


gain_acc = acc_r - acc_not_rot
print("The accuracy gain from rotation is", gain_acc)
```

```
Testing accuracy score for a tree with max depth 2 is 0.956140350877193
The accuracy gain from rotation is 0.02631578947368418
```

Problem 5

Generate samples of 100, 10,000 and 100,000, moons using the code below. Set random seed at 42. Split data in training and testing sets. Estimate separately Logistic, Random Forest, SVC and the hard voting classifier. What happens to the accuracy score as you increase the number of observations? Measyre and report the time it takes for each estimation.

In [85]:

```python
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

In [86]:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = LogisticRegression(random_state=42)
rnd_clf = RandomForestClassifier(random_state=42)
svm_clf = SVC(random_state=42)
# voting classifier has syntaxis akin to pipeline
voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard')
```

In [89]:

```python
from sklearn.metrics import accuracy_score
import time
for N in [100,10000,100000]:
    X, y = make_moons(n_samples=N, noise=0.30, random_state=42)
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
    print(N, "Observations")
    for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
        start = time.time()
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        print(clf.__class__.__name__, accuracy_score(y_test, y_pred), f'Time: {time.time() - start} ')
```

```
100 Observations
LogisticRegression 0.96 Time: 0.0045928955078125
RandomForestClassifier 0.92 Time: 0.14564275741577148
SVC 0.92 Time: 0.0022542476654052734
VotingClassifier 0.96 Time: 0.1466691493988037
10000 Observations
LogisticRegression 0.8588 Time: 0.017252445220947266
RandomForestClassifier 0.9092 Time: 0.8374078273773193
SVC 0.9184 Time: 0.6378459930419922
VotingClassifier 0.9176 Time: 1.4783236980438232
100000 Observations
LogisticRegression 0.85392 Time: 0.12099838256835938
RandomForestClassifier 0.90508 Time: 11.833928108215332
SVC 0.91332 Time: 66.16663932800293
VotingClassifier 0.90976 Time: 78.46461272239685
```

Answer: As we increase the number of observations the quality of prediction using Logistic goes down, which SVC takes the lead. SVC though is very slow. Though generally Voting classifier performs better it is inferior to SVC with the large number of data points.

Problem 6

Generate data using the code provided below. Using testing accuracy as metric, estimate bagging random trees estimator with 200 estimators. Try different numbers of samples: 10, 30, 50, and 200. What is optimal number of samples to be used?
BaggingClassifier(DecisionTreeClassifier(random_state=42), n_estimators=5, max_samples= ?, bootstrap=True, n_jobs=-1, random_state=42)

In [110]:

```python
X, y = make_moons(n_samples=500, noise=0.40, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

In [111]:

```python
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import numpy as np
def bagtree(S):
    bag_clf = BaggingClassifier(
        DecisionTreeClassifier(random_state=42), n_estimators=200,
        max_samples=S, bootstrap=True, n_jobs=-1, random_state=42)
    bag_clf.fit(X_train, y_train)
    y_pred = bag_clf.predict(X_test)
    return accuracy_score(y_test, y_pred)


bag_acc = []

for i in [10, 30, 50,200]:
    acc = bagtree(i)
    vec2 = np.column_stack((i,acc))
    bag_acc.append(vec2)
    print(acc)

print(bag_acc)
```

```
0.76
0.832
0.848
0.832
[array([[10.  ,  0.76]]), array([[30.   ,  0.832]]), array([[50.   ,  0.848]]), array([[200.   ,   0.832]])]
```

Answer

10 samples is clearly not enough, but there is little difference between 100, 300 and 1000 samples.

Problem 7

Find optimal learning rate, number of estimators and maximum depth using GradientBoostingClassifier, and Randomize grid search. Set a grid: learning rate from 0.01 to 3, number of estimators from 1 to 20, and maximum depth from 1 to 10. Try 300 iterations. Example for randomizeSearch:

rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distribs, n_iter=?, cv=5, scoring='neg_mean_squared_error', random_state=42)
gbrt = GradientBoostingClassifier(max_depth=?, n_estimators=?, learning_rate = ?, random_state=42)

Which estimator was the best? What was the accuracy of the best estimator?

In [114]:

```python
X, y = make_moons(n_samples=2000, noise=0.40, random_state=42)
```

In [120]:

```python
import random

from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
from scipy.stats import uniform


grad = GradientBoostingClassifier(random_state=42)
param_distribs = {
        'n_estimators': randint(low=1, high=20),
        'learning_rate': uniform(0.01,3),
        'max_depth': randint(low=1, high=10),
    }
rnd_search = RandomizedSearchCV(grad, param_distributions=param_distribs, n_iter=300, cv=5, scoring='neg_mean_squared_
error', random_state=42)
rnd_search.fit(X, y)
from sklearn.model_selection import cross_val_score
bestest = rnd_search.best_estimator_
scores = np.mean(cross_val_score(bestest, X, y, cv=5))
print("Best estimator", bestest)
print("Produced accuracy",scores)
```

```
Best estimator GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=1.0452137440800489, loss='deviance',
                           max_depth=1, max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=19,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=42, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
Produced accuracy 0.8640000000000001
```

In [ ]: