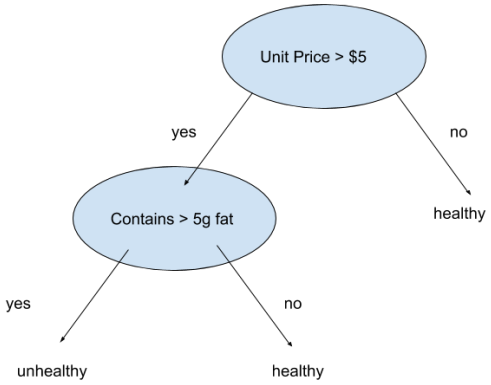
1 Decision Trees

Problem A:



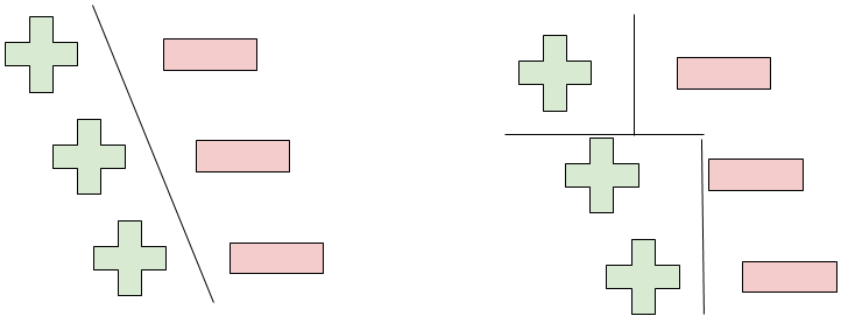
At the first level we will look at the entropy for each category given. Examining 1) package type shows that Bagged generates 2 healthy foods, and Canned generates 1 healthy, 1 unhealthy. This entropy is 2.0. Then, examining 2) unit prices shows that over 5 dollars generates 1 unhealthy and 1 healthy, and under 5 dollars generates 2 healthy. This entropy is 2.0 Finally, looking at 3) fat levels shows that over 5 grams of fat generates 1 unhealthy and 1 healthy, and under 5 grams of fat generates 2 healthy. This entropy is also 2.0.

For the first level, all entropy levels are the same so we can choose to split on unit price.

At the second level we will look at the entropy for remaining categories. Examining 1) package type shows that Bagged generates healthy and Canned generates unhealthy. This entropy is 1.0. Examining 2) fat levels shows that over 5 grams of fat is unhealthy and under 5 grams of fat is healthy. Again, for the second level, all entropy levels are the same so we can choose to split on grams of fat.

We started off with $p_{s'} = .75$ and $S' = 4$ so we calculate our entropy to be about 3.24. Then since our first level had an entropy of 2 we reduce by 1.245. Then moving from the first level to the second level we reduce 2 to end up with an entropy of 0.

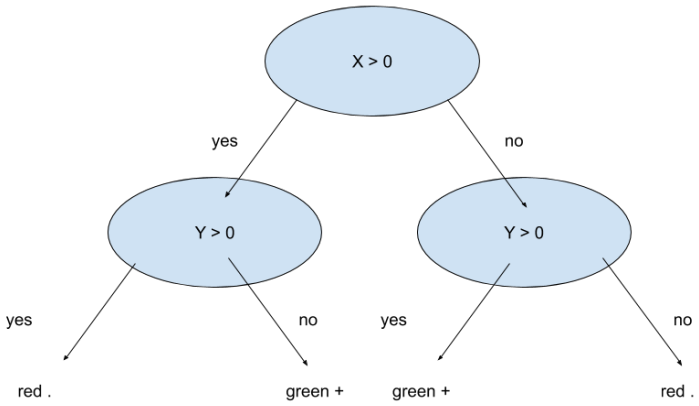
Problem B:
Compared to a linear classifier, a decision tree is not always preferred for classification problems.



Problem C:

i. There is no way to reduce entropy to 0 by splitting the data, so we keep the entropy at 2 and don't split. This means our classification error would be .5 and our tree wouldn't really be a tree, just one node for the whole dataset with an arbitrary prediction since there are half green and half red points.

ii.



We could classify with $(numpos * numpos)/\sqrt{S'}$ where $numpos$ is the number of positive numbers. This would ensure that if there were equal portions of accurate classifications before and after a split, then we would split. But using this measure would also make the model prone to splitting even when not necessary, and generate overfitting.

iii. If we look at our previous problem, with four data points we need 3 splits. And with three data points we would need 2 splits. We see this pattern continues since adding a new point just needs one more split, and if we only have one point, we won't need to split. So

given 100 data points, we'd need 99 splits.

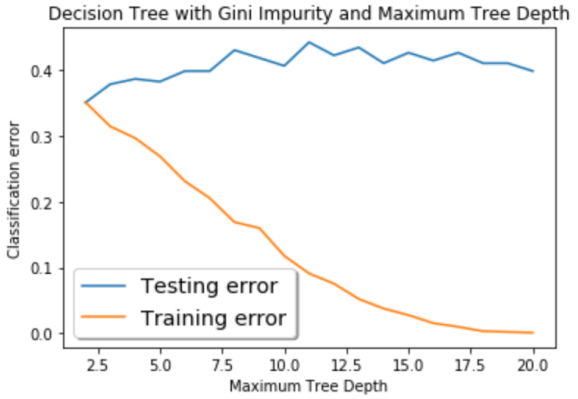
Problem D:
Our worst scenario complexity would be $O(N * D)$ because there are $D * N$ splits and the same number of queries.

2 Overfitting Decision Trees

Problem A:


```
Test error minimized at min_samples_leaf = 12
```

Problem B:




```
Test error minimized at max_depth = 2
```

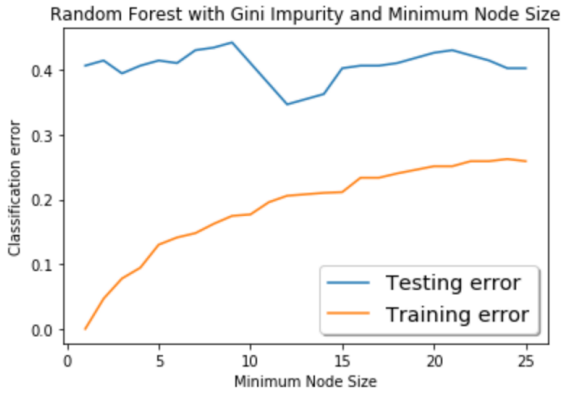
Problem C:

Test error is minimized at max depth of 2 and minimum sample leaf of 12.
Part A tells us that too early could hinder the accuracy of a model, but so could stopping too late. Stopping early gives us low training error but high testing error, but if we stop too late then our training error will be high as well. Early stopping can prevent overfitting and improve generalization but stopping too early can result in high test error.

Part B tells us again that stopping too early could hinder the accuracy of a model, but so could stopping too late. With early stopping in tree depth we see high training error and a

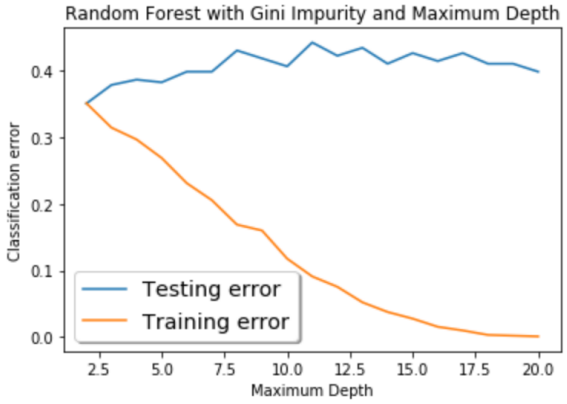
certain increase in testing error up until about a tree depth of 8. Stopping too late has low training error but a slight increase in testing error because of overfitting. We see again that early stopping can prevent overfitting and improve generalization but stopping too early can result in high test error.

Problem D:




```
Test error minimized at min_samples_leaf = 12
```

Problem E:




```
Test error minimized at max_depth = 2
```

Problem F:
Test error is minimized at max depth of 2 and minimum sample leaf of 12.

Part D tells us that too early could hinder the accuracy of a model, but so could stopping too late. Stopping early gives us low training error but high testing error, but if we stop too late then our training error will be high as well. Early stopping can prevent overfitting and

improve generalization but stopping too early can result in high test error.

Part E tells us again that stopping too early could hinder the accuracy of a model, but so could stopping too late. With early stopping in tree depth we see high training error and a certain increase in testing error up until about a tree depth of 8. Stopping too late has low training error but a slight increase in testing error because of overfitting. We see again that early stopping can prevent overfitting and improve generalization but stopping too early can result in high test error.

Problem G:
There seems to be minimal difference between the two graphs.

3 AdaBoost Algorithm

Problem A:
We want to show $E = \frac{1}{N}\sum_{i=1}^{N}\exp(-y_i f(x_i)) \geq \frac{1}{N}\sum_{i=1}^{N} 1(H(x_i) \neq y_i)$.

We know that there are two cases, if y_i and $f(x_i)$ have the same sign, or if they have opposite signs. If they have the same sign then we see that $\exp(-y_i f(x_i))$ should be greater than or equal to 0 since $\exp(-y_i f(x_i)) = \exp(negative\#)$. We see that $1(H(x_i) \neq y_i) = 0$ should also be true in this case, which means that $\exp(-y_i f(x_i))$ must be greater than or equal to $1(H(x_i) \neq y_i)$.

On the other hand, we see that if they have opposite signs then $\exp(-y_i f(x_i))$ should be greater than or equal to 1 since $\exp(-y_i f(x_i)) = \exp(positive\#)$. We see that $1(H(x_i) \neq y_i) = 1$ should also be true in this case, which means that $\exp(-y_i f(x_i))$ must be greater than or equal to $1(H(x_i) \neq y_i)$.

Combining these two together gives us $\exp(-y_i f(x_i))$ must be greater than or equal to $1(H(x_i) \neq y_i)$ in both cases, which effectively proves our point.

Problem B:
We want to find $D_{T+1}(i)$ in terms of Z_t, α_t, x_i, y_i. From lecture we know that $D_{t+1}(i) = \frac{D_t(i)\exp(-a_t y_i h_t(x_i))}{Z_t}$.

Problem C & D:
We want to show that $E = \prod_{t=1}^{T} Z_t$, which we can do using the formula from the previous problem: $D_{t+1}(i) = \frac{D_t(i)\exp(-a_t y_i h_t(x_i))}{Z_t}$ This means that $D_t(i)$ must be the same as $D_1(i)$ times $(\prod_{t=1}^{t-1}\frac{D_t(i)\exp(-a_t y_i h_t(x_i))}{Z_t})$.

We know that $Z_t = \sum_{i=1}^{N} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$. Since we know $D_t(i)$ we can plug in to get $Z_t = \sum_{i=1}^{N}(\prod_{t=1}^{t-1}\frac{D_t(i)\exp(-a_t y_i h_t(x_i))}{Z_t})D_1(i)\exp(-\alpha_t y_i h_t(x_i))$ and manipulate this equation to get $Z_t = \sum_{i=1}^{N}(\prod_{t=1}^{t-1}\frac{1}{Z_t})(\prod_{t=1}^{t-1}\exp(-a_t y_i h_t(x_i)))D_1 i$.

From lecture 6 we know that $\exp(-y_i\sum_{t=1}^{n}\alpha_t h_t(x_i)) = \left(\prod_{t=1}^{n}\exp(-y a_t h_t(x_i))\right)$. We use substitution to derive $Z_t = \sum_{i=1}^{N}(\prod_{t=1}^{t-1}\frac{1}{Z_t})(\exp(-y_i\sum_{t=1}^{n}\alpha_t h_t(x_i)))D_1 i$, which is the same as $Z_t = \sum_{i=1}^{N}(\prod_{t=1}^{t-1}\frac{1}{Z_t})(\exp(-y_i f(x_i)))D_1 i$ and $(\prod_{t=1}^{t-1} Z_t)Z_t = \sum_{i=1}^{N}(\exp(-y_i f(x_i)))D_1 i$ and fi-

nally $\prod_{t=1}^{t} Z_t = \sum_{i=1}^{N}(\exp(-y_i f(x_i)))D_1 i$. Since we know that $D_1(i) = \frac{1}{N}$, we see that $E = \sum_{i=1}^{N} \frac{1}{N}(\exp(-y_i f(x_i)))$

Problem E & F:
We first want to show $Z_t = (1-\epsilon_t)\exp(-\alpha_t) + \epsilon_t \exp(\alpha_t)$, which we can do by considering when $h_t(x_i) \neq y_i$ and when it is. In the first possibility, if we manipulate $Z_t = \sum_{i=1}^{N}(D_t(i)\exp(-a_t y_i h_t(x_i))$ we can get $Z_t = \sum_{i=1}^{N}(D_t(i)\exp(a_t) = (\exp(a_t))$. We know $\epsilon_t = \sum_{i=1}^{N}(D_t(i)1(h_t(x_i) \neq y_i)) = 1$ so $Z_t = (1-\epsilon_t)\exp(-\alpha_t)+\epsilon_t\exp(\alpha_t) = (0)\exp(-\alpha_t)+1\exp(\alpha_t) = \exp(\alpha_t)$. If $h_t(x_i) = y_i$ then we can say again $Z_t = \sum_{i=1}^{N}(D_t(i)\exp(-a_t y_i h_t(x_i))$. This equals $Z_t = \sum_{i=1}^{N}(D_t(i)\exp(-\alpha_t) = \exp(-\alpha_t)$. If we use the equation we want to show, we know that $\epsilon_t = 0$ so we get $Z_t = (1-\epsilon_t)\exp(-\alpha_t) + \epsilon_t\exp(\alpha_t) = Z_t = (1)\exp(-\alpha_t) + 0\exp(\alpha_t) = \exp(-\alpha_t)$.

Now, knowing $Z_t = (1-\epsilon_t)\exp(-\alpha_t) + \epsilon_t\exp(\alpha_t)$, we can take the derivative with respect to α_t to get

$$
\begin{aligned}
0 &= (1-\epsilon_t)(-\alpha_t)\exp(-\alpha_t) + \epsilon_t\alpha_t\exp(\alpha_t) \\
\epsilon_t\alpha_t\exp(\alpha_t) &= \alpha_t(1-\epsilon_t)\exp(-\alpha_t) \\
\alpha_t\exp(2\alpha_t) &= \alpha_t\frac{1-\epsilon_t}{\epsilon_t} \\
\exp(2\alpha_t) &= \frac{1-\epsilon_t}{\epsilon_t} \\
\alpha_t &= \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}
\end{aligned}
$$

Problem G—J:
I couldn't get my code to work in time :(sorry!

Problem 2

Import statements and function to load data:

In [50]:
```
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import numpy as np

# Seed the random number generator:
np.random.seed(1)

def load_data(filename, skiprows = 1):
    """
    Function loads data stored in the file filename and returns it as a numpy ndar
ray.

    Inputs:
        filename: given as a string.

    Outputs:
        Data contained in the file, returned as a numpy ndarray
    """
    return np.loadtxt(filename, skiprows=skiprows, delimiter=',')
```

Load the data and divide it into training and validation sets:

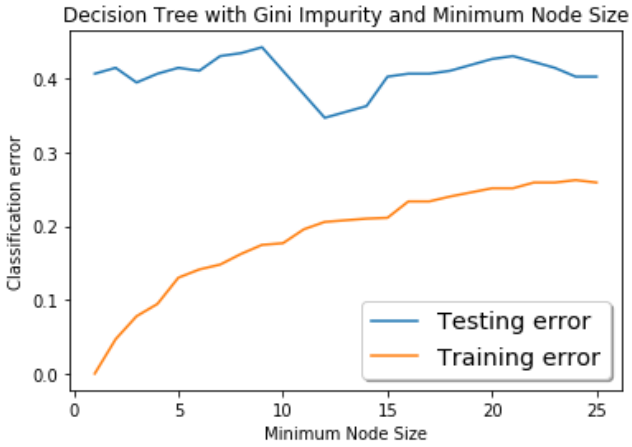
In [51]:
```
# The number 24 in the next line corresponds to the number of header lines
X = load_data('data/messidor_features.arff', 24)

data = X[:, :-1]
diag = X[:, -1]

train_size = 900

train_data = data[0:train_size]
train_label = diag[0:train_size]
test_data = data[train_size:]
test_label = diag[train_size:]
```

Problem 2A: Decision Trees with Minimum Leaf Size Stopping Criterion

Fill in the two functions below:

In [52]:
```
def classification_err(y, real_y):
    """
    This function returns the classification error between two equally-sized vecto
rs of
    labels; this is the fraction of samples for which the labels differ.

    Inputs:
        y: (N, ) shaped array of predicted labels
        real_y: (N, ) shaped array of true labels
    Output:
        Scalar classification error
    """
    #============================================
    # TODO: Implement the classification_err function,
    # based on the above instructions.
    #============================================
    err = 0
    for i in range(len(y)):
        if y[i] != real_y[i]:
            err += 1
    return err / len(y)

def eval_tree_based_model_min_samples(clf, min_samples_leaf, X_train, y_train, X_t
est, y_test):
    """
    This function evaluates the given classifier (either a decision tree or random
forest) at all of the
    minimum leaf size parameters in the vector min_samples_leaf, using the given t
raining and testing
    data. It returns two vector, with the training and testing classification erro
rs.

    Inputs:
        clf: either a decision tree or random forest classifier object
        min_samples_leaf: a (T, ) vector of all the min_samples_leaf stopping cond
ition parameters
                            to test, where T is the number of parameters to test
        X_train: (N, D) matrix of training samples.
        y_train: (N, ) vector of training labels.
        X_test: (N, D) matrix of test samples
        y_test: (N, ) vector of test labels
    Output:
        train_err: (T, ) vector of classification errors on the training data
        test_err: (T, ) vector of classification errors on the test data
    """
    #============================================================
    # TODO: Implement the eval_tree_based_model_min_samples function,
    # based on the above instructions.
    #============================================================
    train_err = []
    test_err = []

    for m in min_samples_leaf:
        clf = tree.DecisionTreeClassifier(criterion='gini',min_samples_leaf=m)

        clf.fit(X_train, y_train)

        trainPredicted = clf.predict(X_train)
        testPredicted = clf.predict(X_test)

        train_err.append(classification_err(trainPredicted, y_train))
        test_err.append(classification_err(testPredicted, y_test))

    return train_err, test_err
```

In [53]:
```
# Seed the random number generator:
np.random.seed(1)

min_samples_leaf = np.arange(1, 26)
clf = tree.DecisionTreeClassifier(criterion='gini')

train_err, test_err = eval_tree_based_model_min_samples(clf, min_samples_leaf, tra
in_data, train_label, test_data, test_label)

plt.figure()
plt.plot(min_samples_leaf, test_err, label='Testing error')
plt.plot(min_samples_leaf, train_err, label='Training error')
plt.xlabel('Minimum Node Size')
plt.ylabel('Classification error')
plt.title('Decision Tree with Gini Impurity and Minimum Node Size')
plt.legend(loc=0, shadow=True, fontsize='x-large')
plt.show()

print('Test error minimized at min_samples_leaf = %i' % min_samples_leaf[np.argmin
(test_err)])
```




```
Test error minimized at min_samples_leaf = 12
```

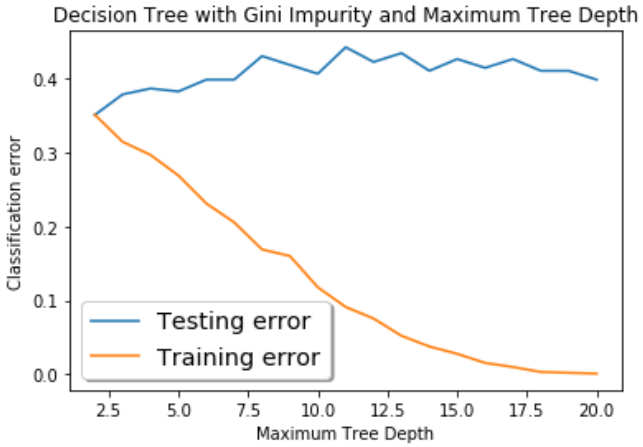
Problem 2B: Decision Trees with Maximum Depth Stopping Criterion

Fill in the function below:

In [54]:
```
def eval_tree_based_model_max_depth(clf, max_depth, X_train, y_train, X_test, y_te
st):
    """
    This function evaluates the given classifier (either a decision tree or random
forest) at all of the
    maximum tree depth parameters in the vector max_depth, using the given trainin
g and testing
    data. It returns two vector, with the training and testing classification erro
rs.

    Inputs:
        clf: either a decision tree or random forest classifier object
        max_depth: a (T, ) vector of all the max_depth stopping condition paramete
rs
                            to test, where T is the number of parameters to test
        X_train: (N, D) matrix of training samples.
        y_train: (N, ) vector of training labels.
        X_test: (N, D) matrix of test samples
        y_test: (N, ) vector of test labels
    Output:
        train_err: (T, ) vector of classification errors on the training data
        test_err: (T, ) vector of classification errors on the test data
    """
    #====================================================================
    # TODO: Implement the eval_tree_based_model_max_depth function,
    # based on the above instructions.
    #====================================================================
    train_err = []
    test_err = []

    for d in max_depth:
        clf = tree.DecisionTreeClassifier(criterion='gini',max_depth=d)

        clf.fit(X_train, y_train)

        trainPredicted = clf.predict(X_train)
        testPredicted = clf.predict(X_test)

        train_err.append(classification_err(trainPredicted, y_train))
        test_err.append(classification_err(testPredicted, y_test))

    return train_err, test_err
```

In [ ]:
```

```

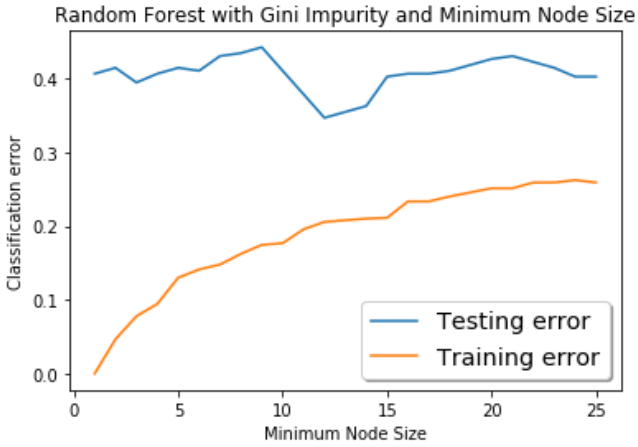
In [57]:
```
# Seed the random number generator:
np.random.seed(1)

max_depth = np.arange(2, 21)

clf = tree.DecisionTreeClassifier(criterion='gini')

train_err, test_err = eval_tree_based_model_max_depth(clf, max_depth, train_data,
                                                      train_label, test_data, te
st_label)

plt.figure()
plt.plot(max_depth, test_err, label='Testing error')
plt.plot(max_depth, train_err, label='Training error')
plt.xlabel('Maximum Tree Depth')
plt.ylabel('Classification error')
plt.title('Decision Tree with Gini Impurity and Maximum Tree Depth')
plt.legend(loc=0, shadow=True, fontsize='x-large')
plt.show()

print('Test error minimized at max_depth = %i' % max_depth[np.argmin(test_err)])
```




```
Test error minimized at max_depth = 2
```

Problem 2D: Random Forests with Minimum Leaf Size Stopping Criterion

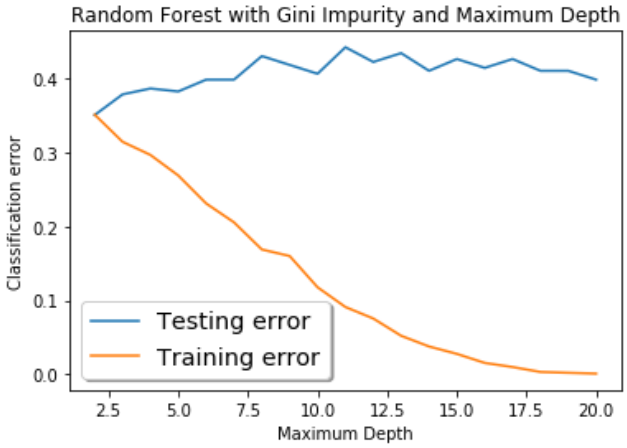
In [55]:
```
# Seed the random number generator:
np.random.seed(1)

n_estimators = 1000
clf = RandomForestClassifier(n_estimators = n_estimators, criterion = 'gini')

min_samples_leaf = np.arange(1, 26)

train_err, test_err = eval_tree_based_model_min_samples(clf, min_samples_leaf, tra
in_data,
                                                        train_label, test_data, te
st_label)

plt.figure()
plt.plot(min_samples_leaf, test_err, label='Testing error')
plt.plot(min_samples_leaf, train_err, label='Training error')
plt.xlabel('Minimum Node Size')
plt.ylabel('Classification error')
plt.title('Random Forest with Gini Impurity and Minimum Node Size')
plt.legend(loc=0, shadow=True, fontsize='x-large')
plt.show()

print('Test error minimized at min_samples_leaf = %i' % min_samples_leaf[np.argmin
(test_err)])
```




```
Test error minimized at min_samples_leaf = 12
```

Problem 2E: Random Forests with Maximum Depth Stopping Criterion

In [56]:
```
# Seed the random number generator:
np.random.seed(1)

clf = RandomForestClassifier(n_estimators = n_estimators, criterion = 'gini')

max_depth = np.arange(2, 21)

train_err, test_err = eval_tree_based_model_max_depth(clf, max_depth, train_data,
                                                      train_label, test_data, te
st_label)

plt.figure()
plt.plot(max_depth, test_err, label='Testing error')
plt.plot(max_depth, train_err, label='Training error')
plt.xlabel('Maximum Depth')
plt.ylabel('Classification error')
plt.title('Random Forest with Gini Impurity and Maximum Depth')
plt.legend(loc=0, shadow=True, fontsize='x-large')
plt.show()

print('Test error minimized at max_depth = %i' % max_depth[np.argmin(test_err)])
```




```
Test error minimized at max_depth = 2
```

In [ ]:

In [ ]: