Decision Trees - a decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g., whether a car's horsepower is larger than 200), each branch represents the outcome of the test, and each leaf node represents an outcome (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

In [42]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf

from sklearn.linear_model import LogisticRegression
from tqdm import tqdm
from urllib.error import URLError

sns.set(font_scale=1.5)
sns.set_style("whitegrid", {'grid.linestyle':'--'})
```

In [43]:
```
# import the breast cancer dataset
cancer = pd.read_csv("https://raw.githubusercontent.com/changyaochen/MECE4520/master/data/breast_
cancer["label"] = cancer["diagnosis"].apply(lambda x: 0 if x == "B" else 1)     # creates a new c
cancer
```

Out[43]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concav
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	
...	...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	

569 rows × 33 columns

In [44]:
```
# Problem 1 - binary classification
```

In [45]:
```
# what is the value of the gini index for the full dataset

cancer["label"] = cancer["diagnosis"].apply(lambda x: 0 if x == "B" else 1) # 0 if benign, 1 if m

def calculate_gini_index(labels):       # labels is based off the the diagnosis column
    total_samples = len(labels)         # total amount of diagnosis
    if total_samples == 0:              # if all samples are benign (label of 0)
        return 0                        # return 0 if all samples are benign (0)

    class_1_probability = labels.sum() / total_samples    # ratio of malignant samples (label of
    class_0_probability = 1 - class_1_probability         # probability of benign samples (label

    gini_index = 1 - (class_1_probability ** 2 + class_0_probability ** 2)     # calculate the gi
    return gini_index                                                          # return gini inde

gini_index_cancer = calculate_gini_index(cancer["label"])
print("The Gini Index for the entire Breast Cancer Dataset is", gini_index_cancer)
```

```
The Gini Index for the entire Breast Cancer Dataset is 0.4675300607546925
```

In [46]:
```
# use gini index as the loss metric, split the feature of concave_extreme, what is the value of t

# import the different features
from sklearn import tree

dt_model = tree.DecisionTreeClassifier(
    criterion = "gini",
    max_depth = 1,
)
features = [
    #"radius_mean",
    #"texture_mean",
    #"perimeter_mean",
    #"area_mean",
    #"smoothness_mean",
    #"compactness_mean",
```

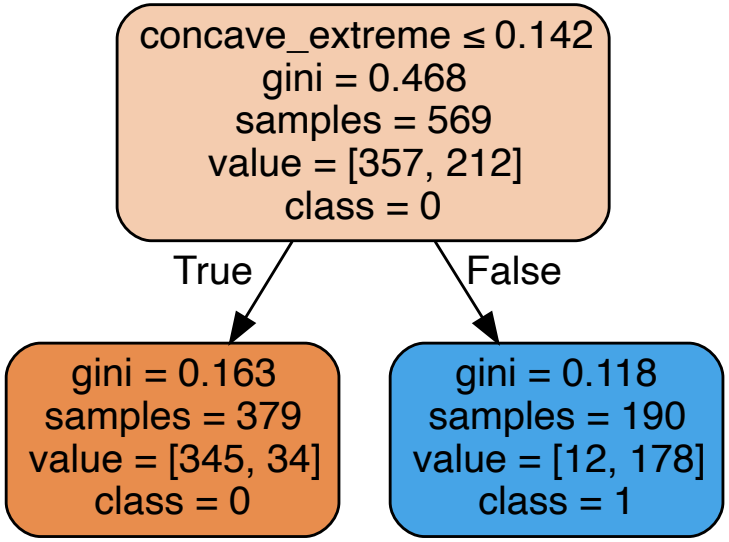
```
    #"concavity_mean",
    #"concave_mean",
    #"symmetry_mean",
    #"fractal_mean",
    #"radius_se",
    #"texture_se",
    #"perimeter_se",
    #"area_se",
    #"smoothness_se",
    #"compactness_se",
    #"concavity_se",
    #"concave_se",
    #"symmetry_se",
    #"fractal_se",
    #"radius_extreme",
    #"texture_extreme",
    #"perimeter_extreme",
    #"area_extreme",
    #"smoothness_extreme",
    #"compactness_extreme",
    #"concavity_extreme",
    "concave_extreme",              # want the concave_extreme feature
    #"symmetry_extreme",
    #"fractal_extreme",
]

label = "label"
dt_model.fit(X = cancer[features], y = cancer[label])
```

Out[46]:

```
▾      DecisionTreeClassifier
DecisionTreeClassifier(max_depth=1)
```

In [47]:
```
import graphviz    # creating decision tree graphic - from professor's github

dot_data = tree.export_graphviz(     # use graphviz library to create the decision tree graphic
    decision_tree = dt_model,        # model for decision tree - block above
    out_file = None,
    feature_names = features,        # which features to use - block above dictates
    class_names = ["0", "1"],        # benign or malignant
    filled = True,
    rounded = True,
    special_characters = True,
    max_depth = 1,
)

graph = graphviz.Source(dot_data)    # define the graph
graph.render("cancer_tree")
graph                                # print the graph
```

Out[47]:



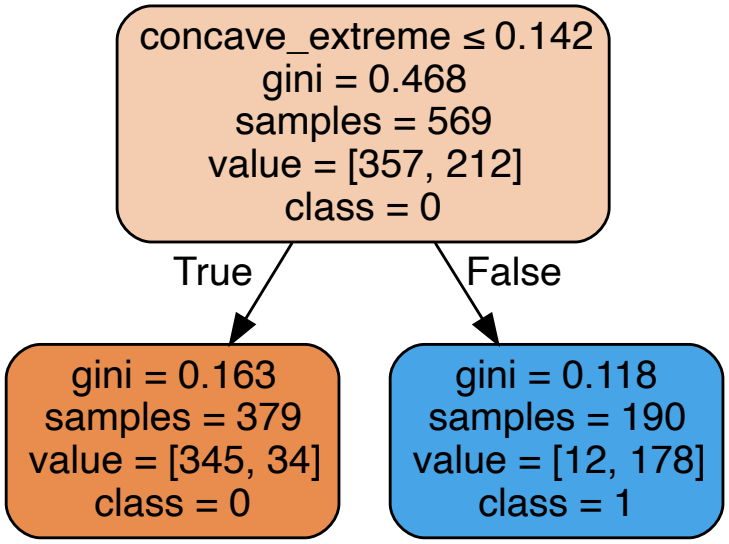
In [48]:

```
# 0 = benign = True
# 1 = malignant = False

# find the value of the cutpoint using general code for cutpoint - use again when solving with er
def calculate_best_split(data, feature_name):       # define the best split point function
    unique_values = data[feature_name].unique()     # unique values of specified feature
    best_gini_index = float('inf')                  # best gini index - initialized to positive i
    best_cut_point = None                           # best cut point - initialized to none

    for value in unique_values:                                 # begins iteration through each uniqu
        left_subset = data[data[feature_name] <= value]     # subset of the data that is less tha
        right_subset = data[data[feature_name] > value]     # subset of the data that is greater

        gini_left = calculate_gini_index(left_subset["label"])      # calculate left gini index
        gini_right = calculate_gini_index(right_subset["label"])    # calculate right gini index

        total_samples = len(data)                               # number of total breast cancer sampl
        weight_left = len(left_subset) / total_samples      # weight of the left subset based on
        weight_right = len(right_subset) / total_samples    # weight of the right subset based or

        gini_index_split = weight_left * gini_left + weight_right * gini_right     # calculates t

        if gini_index_split < best_gini_index:      # check if current split has a lower gini ind
            best_gini_index = gini_index_split      # if true, set a new best gini index and cutp
            best_cut_point = value                  # best cut point equals the value correspondi

    return best_cut_point                           # return the best cut point

# apply it to concave_extreme
data = cancer
feature_name = "concave_extreme"
best_cut_point = calculate_best_split(data, feature_name)
print(f"The best value of the cut point when using gini index as the loss function is {best_cut_p
```

```
The best value of the cut point when using gini index as the loss function is 0.1423
```

In [49]:
```
# with this cut point, how many samples are in each of the two resulting partitions
graph     # note samples section on the cut point
```

Out[49]:



379 benign samples 190 malignant samples

In [50]:
```
# repeat these steps using entropy instead of gini index
```

In [51]:
```
# what is the value of entropy for the full dataset

cancer["label"] = cancer["diagnosis"].apply(lambda x: 0 if x == "B" else 1)

def calculate_entropy(labels):          # labels is based off the the diagnosis column # define t
    total_samples = len(labels)         # total amount of diagnosis/samples
    if total_samples == 0:              # if all samples are benign (label of 0)
        return 0                        # return 0 if all samples are benign

    class_1_probability = labels.sum() / total_samples    # ratio of malignant samples (label of
    class_0_probability = 1 - class_1_probability         # probability of benign samples (label

    entropy = - (class_1_probability * np.log2(class_1_probability + 1e-10) + class_0_probability
    return entropy

entropy_cancer = calculate_entropy(cancer["label"])
print("The Entropy for the entire Breast Cancer Dataset is", entropy_cancer)
```

```
The Entropy for the entire Breast Cancer Dataset is 0.9526351221133209
```

In [52]:
```
# use entropy as the loss metric, split the feature of concave_extreme, what is the value of the

# import the different features
from sklearn import tree

dt_model = tree.DecisionTreeClassifier(
    criterion = "entropy",
    max_depth = 1,
)
features = [
    #"radius_mean",
    #"texture_mean",
    #"perimeter_mean",
    #"area_mean",
    #"smoothness_mean",
    #"compactness_mean",
```

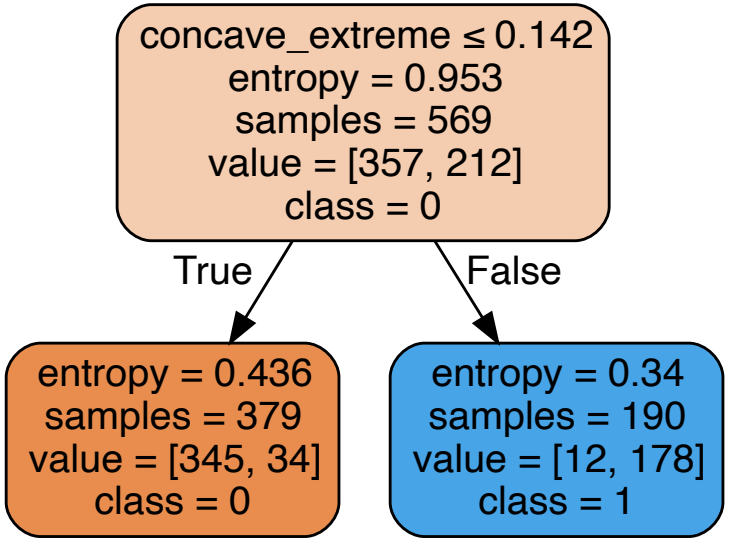
```
    #"concavity_mean",
    #"concave_mean",
    #"symmetry_mean",
    #"fractal_mean",
    #"radius_se",
    #"texture_se",
    #"perimeter_se",
    #"area_se",
    #"smoothness_se",
    #"compactness_se",
    #"concavity_se",
    #"concave_se",
    #"symmetry_se",
    #"fractal_se",
    #"radius_extreme",
    #"texture_extreme",
    #"perimeter_extreme",
    #"area_extreme",
    #"smoothness_extreme",
    #"compactness_extreme",
    #"concavity_extreme",
    "concave_extreme",              # want the concave_extreme feature
    #"symmetry_extreme",
    #"fractal_extreme",
]

label = "label"
dt_model.fit(X = cancer[features], y = cancer[label])
```

Out[52]:

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=1)
```

In [53]:
```
import graphviz    # creating decision tree graphic - from professor's github

dot_data = tree.export_graphviz(     # use graphviz library to create the decision tree graphic
    decision_tree = dt_model,        # model for decision tree - block above
    out_file = None,
    feature_names = features,        # which features to use - block above dictates
    class_names = ["0", "1"],        # benign or malignant
    filled = True,
    rounded = True,
    special_characters = True,
    max_depth = 1,
)

graph = graphviz.Source(dot_data)    # define the graph
graph.render("cancer_tree")
graph                                # print the graph
```

Out[53]:



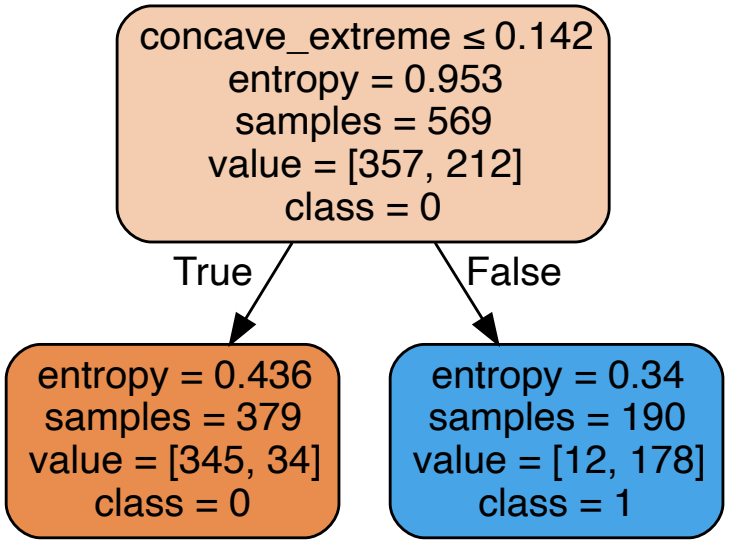
In [54]:

```
# 0 = benign = True
# 1 = malignant = False

# find the value of the cutpoint using general code for cutpoint - replace gini with entropy
def calculate_best_split(data, feature_name):       # define the best split point function
    unique_values = data[feature_name].unique()     # unique values of specified feature (in this
    best_entropy = float('inf')                     # best entropy - initialized to positive infi
    best_cut_point = None                           # best cut point - initialized to none

    for value in unique_values:                             # begins iteration through each uniqu
        left_subset = data[data[feature_name] <= value]     # subset of the data that is less tha
        right_subset = data[data[feature_name] > value]     # subset of the data that is greater

        entropy_left = calculate_entropy(left_subset["label"])      # calculate left entropy - c
        entropy_right = calculate_entropy(right_subset["label"])    # calculate right entropy -

        total_samples = len(data)                           # number of total samples/diagnosis
        weight_left = len(left_subset) / total_samples      # weight of the left subset based on
        weight_right = len(right_subset) / total_samples    # weight of the right subset based or

        entropy_split = weight_left * entropy_left + weight_right * entropy_right     # calculate

        if entropy_split < best_entropy:            # check if current split has a lower entropy
            best_entropy = entropy_split            # if true, set a new best entropy and cutpoir
            best_cut_point = value                  # best cut point equals the value correspondi

    return best_cut_point                           # return the best cut point

# apply it to concave_extreme
data = cancer
feature_name = "concave_extreme"
best_cut_point = calculate_best_split(data, feature_name)
print(f"The best value of the cut point when using entropy as the loss function is {best_cut_poir
```

```
The best value of the cut point when using entropy as the loss function is 0.1423
```

In [55]:
```
# with this cut point, how many samples are in each of the two resulting partitions
graph
```

Out[55]:



379 benign samples 190 malignant samples

Problem 2 - bootstrap sample - obtain a bootstrap sample from a set of n observations

Bootstraping is a resampling technique used to estimate the distribution of a statistic (like the mean or variance) by repeatedly resampling with replacement from the data set. It's a powerful tool used in inferential statistics and is particularly useful when the sample size is small or when the underlying distribution of the data is unknown or complex.

What is the probability that the first bootstrap observation is not the j-th observation from the original sample?

```
probability = 1 - 1/n

this is the probability of not selecting the j-th observation
```

What is the probability that the j-th observation is not in the full bootstrap sample?

```
probability = (1 - 1/n)**b

this is the probability
```

When n = 10000, what is the numerical value of the probability mentioned in section 2?

```
n = 10000
probability = (1 - 1/n)**b

probability = p = (1 - 1/10000)**b
p = (0.9999)**b
p = 0.9999
```