Classification & Regression Trees

Justin Post

Recap

- Determine if we are doing a prediction or classification problem
- Given a model, we **fit** the model using data via a loss function
- Must determine how well the model predicts on **new** data (or using CV) via a metric

Multiple Linear Regression

- Commonly used model with a numeric response

Logistic Regression

- Commonly used model with a binary response

Regression/Classification Trees

Tree based method:

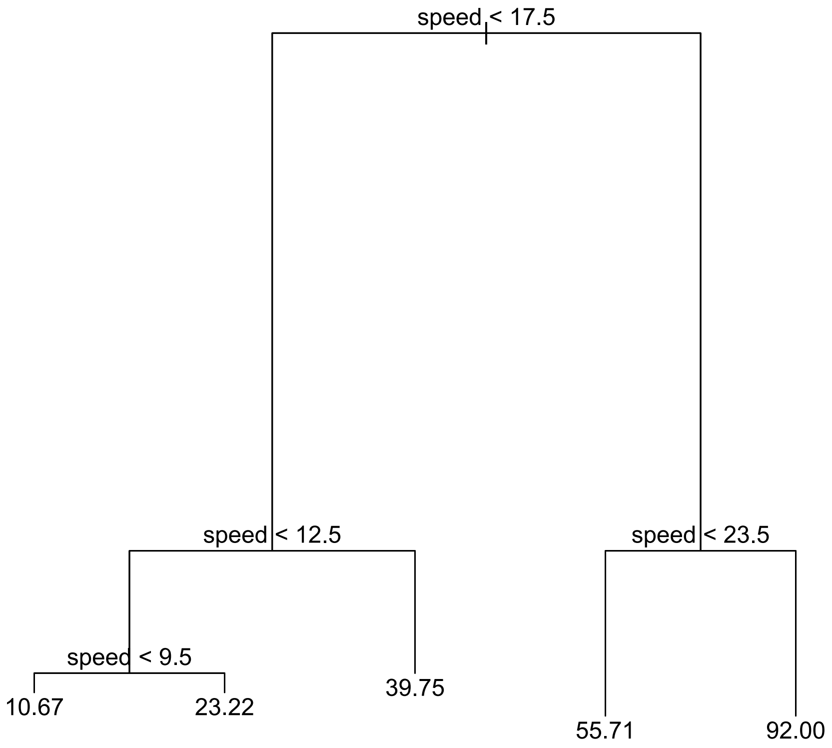
- **Split up predictor space into regions**, different predictions for each region
- *Classification* tree if goal is to classify (predict) group membership
    - Usually use **most prevalent class** in region as prediction

Regression/Classification Trees

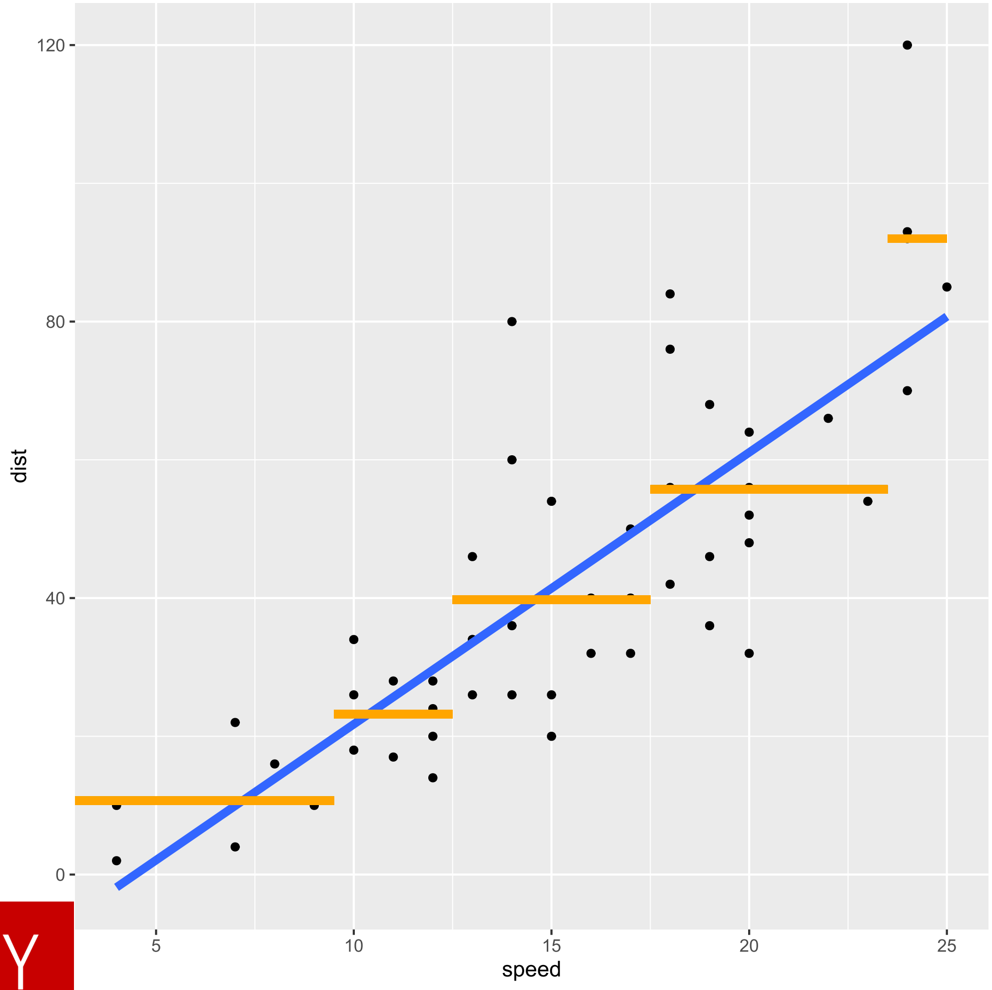
Tree based method:

- **Split up predictor space into regions**, different predictions for each region
- *Classification* tree if goal is to classify (predict) group membership
    - Usually use **most prevalent class** in region as prediction
- *Regression* tree if goal is to predict a continuous response
    - Usually use **mean of observations** in region as prediction

Easy Interpretation



Predictor Space Split vs Linear Function



Fit Regression Tree

- Recall the Bike data and `log_selling_price` as our response

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
bike_data = pd.read_csv("data/bikeDetails.csv")
#create response and new predictor
bike_data['log_selling_price'] = np.log(bike_data['selling_price'])
bike_data['log_km_driven'] = np.log(bike_data['km_driven'])
```

Fit Regression Tree

- Code modified from the docs
- Depth represents how many levels of splits to do

```
from sklearn.tree import DecisionTreeRegressor
regr1 = DecisionTreeRegressor(max_depth=2)
regr2 = DecisionTreeRegressor(max_depth=5)
regr1.fit(bike_data['log_km_driven'].values.reshape(-1,1), bike_data['log_selling_price'].values)
regr2.fit(bike_data['log_km_driven'].values.reshape(-1,1), bike_data['log_selling_price'].values)
```

Fit Regression Tree

- Code modified from the docs
- Depth represents how many levels of splits to do

```
from sklearn.tree import DecisionTreeRegressor
regr1 = DecisionTreeRegressor(max_depth=2)
regr2 = DecisionTreeRegressor(max_depth=5)
regr1.fit(bike_data['log_km_driven'].values.reshape(-1,1), bike_data['log_selling_price'].values)
regr2.fit(bike_data['log_km_driven'].values.reshape(-1,1), bike_data['log_selling_price'].values)
```

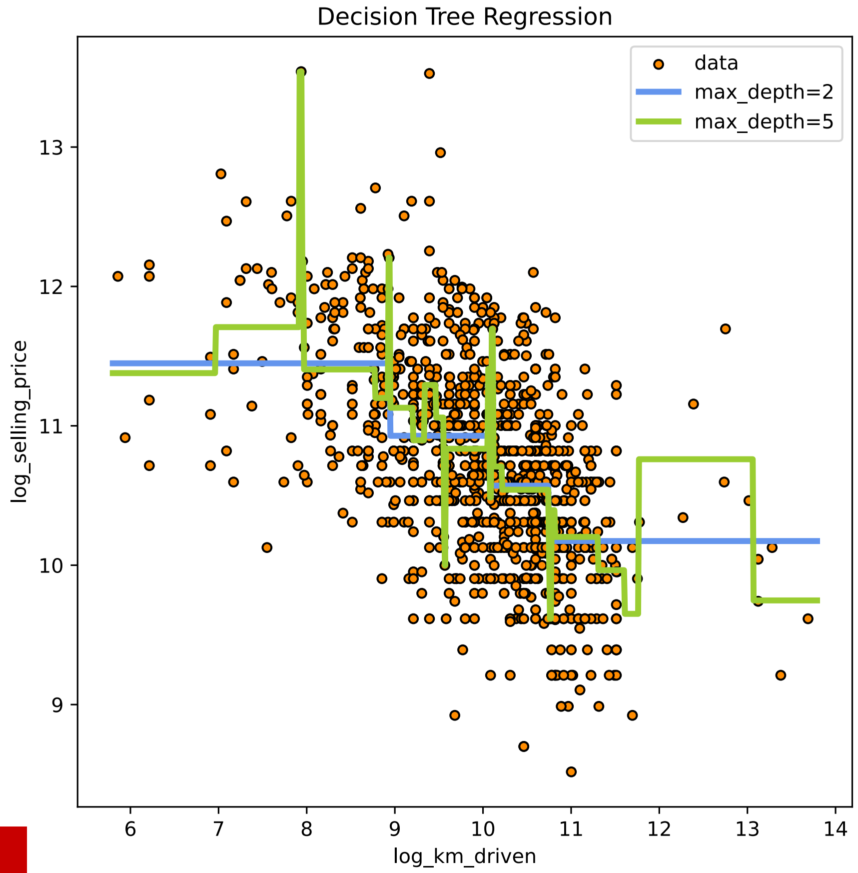
- Can still predict with `.predict()` method

```
X_test = np.arange(5.8, 13.8, 0.01)[:, np.newaxis]
pred1 = regr1.predict(X_test)
pred2 = regr2.predict(X_test)
```

Fit Regression Tree

```
plt.scatter(bike_data['log_km_driven'], bike_data['log_selling_price'],
            s=20, edgecolor="black", c="darkorange", label="data")
plt.plot(X_test, pred1, color="cornflowerblue", label="max_depth=2", linewidth=3)
plt.plot(X_test, pred2, color="yellowgreen", label="max_depth=5", linewidth=3)
plt.xlabel("log_km_driven")
plt.ylabel("log_selling_price")
plt.title("Decision Tree Regression")
plt.legend()
plt.show()
```

Fit Regression Tree



NC STATE UNIVERSITY

Regression Trees

- Extension to more than one predictor is exactly the same idea
- Instead of fitting a plane or saddle in MLR, fit a series of flat planes (mean for a given region)

Regression Trees

- Extension to more than one predictor is exactly the same idea
- Instead of fitting a plane or saddle in MLR, fit a series of flat planes (mean for a given region)

```
from sklearn.tree import DecisionTreeRegressor
regr3 = DecisionTreeRegressor(max_depth=2)
regr3.fit(bike_data[['log_km_driven', 'year']].values, bike_data['log_selling_price'].values)
```

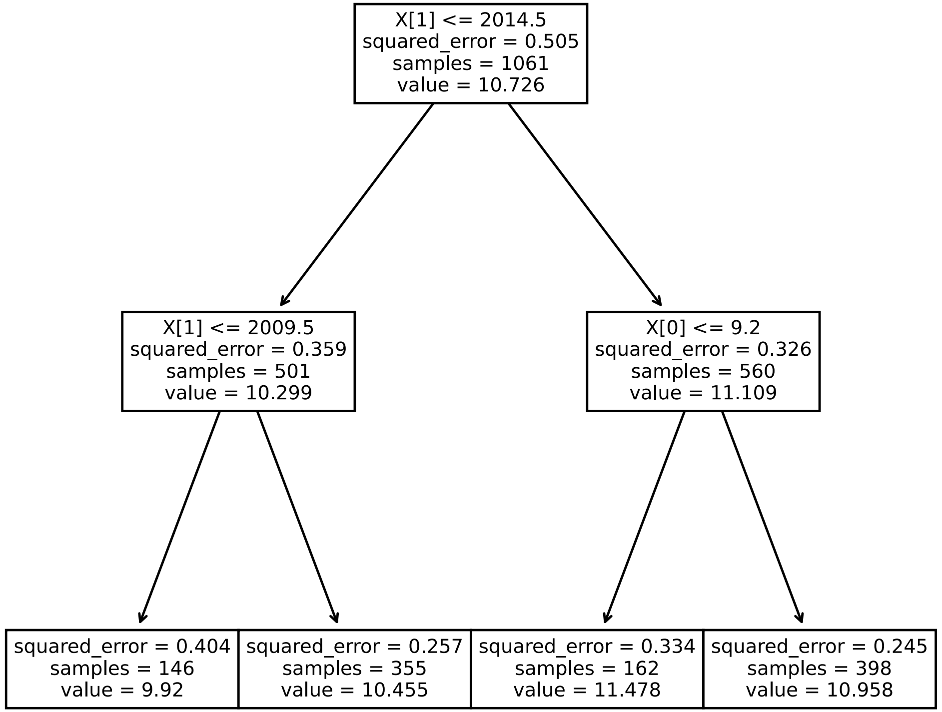
Regression Trees

- Extension to more than one predictor is exactly the same idea
- Instead of fitting a plane or saddle in MLR, fit a series of flat planes (mean for a given region)

```
from sklearn.tree import DecisionTreeRegressor
regr3 = DecisionTreeRegressor(max_depth=2)
regr3.fit(bike_data[['log_km_driven', 'year']].values, bike_data['log_selling_price'].values)
```

- `plot_tree()` function useful for visualization

```
from sklearn.tree import plot_tree
plot_tree(regr3)
```



NC STATE UNIVERSITY

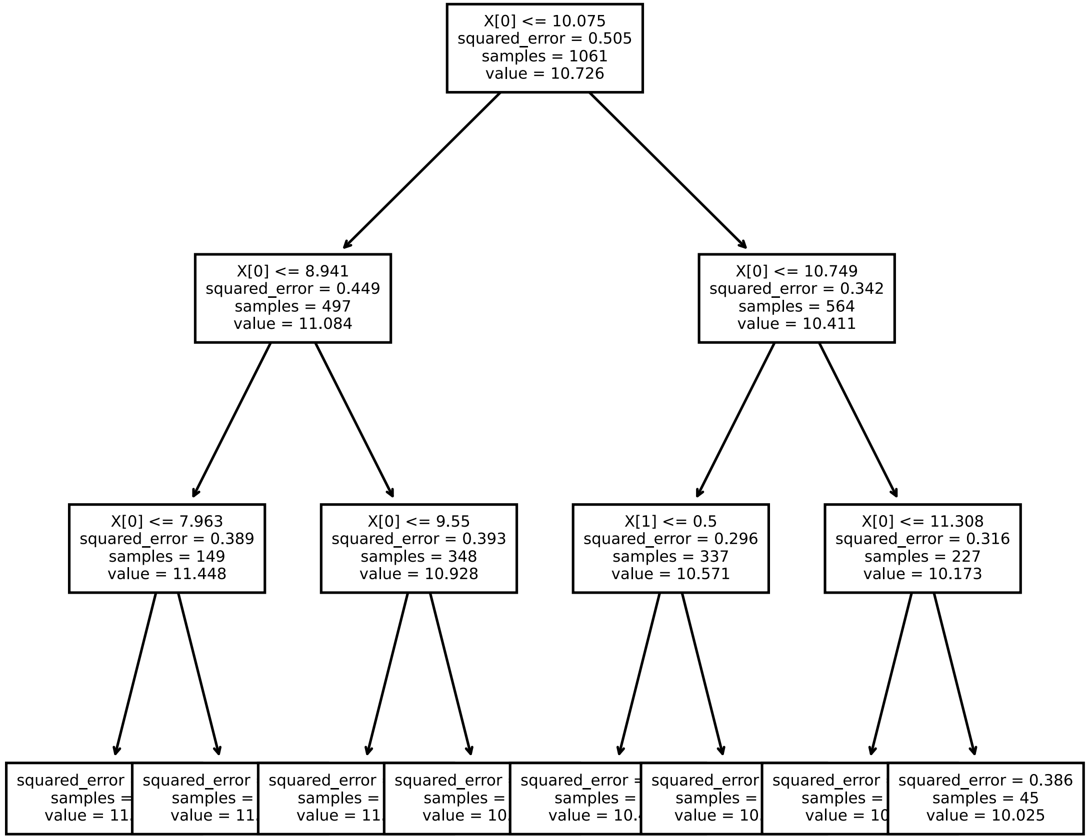
No Need for Interaction Terms

- Trees automatically account for interactions
- An interaction implies that the **effect** of one variable differs depending on the value of another
    - Splitting on more than one variable implies this is the case!

Categorical Predictors

- Easy to include as well
- Must convert to dummy variables though

```
regr4 = DecisionTreeRegressor(max_depth=3)
pd.get_dummies(bike_data.owner).head()
bike_data["owners"] = pd.get_dummies(bike_data.owner)['1st owner']
regr4.fit(bike_data[['log_km_driven', 'owners']].values, bike_data['log_selling_price'].values)
```



NC STATE UNIVERSITY

Regression/Classification Trees

Tree based method:

- **Split up predictor space into regions**, different predictions for each region
- *Classification* tree if goal is to classify (predict) group membership
  - Usually use **most prevalent class** in region as prediction
- *Regression* tree if goal is to predict a continuous response
  - Usually use **mean of observations** in region as prediction

Classification Tree

- Recall the water potability data and `Potability` as our response

```
water = pd.read_csv("data/water_potability.csv")
water.head()
```

```
##          ph    Hardness        Solids  ...  Trihalomethanes  Turbidity  Potability
## 0       NaN  204.890455  20791.318981  ...        86.990970   2.963135           0
## 1  3.716080  129.422921  18630.057858  ...        56.329076   4.500656           0
## 2  8.099124  224.236259  19909.541732  ...        66.420093   3.055934           0
## 3  8.316766  214.373394  22018.417441  ...       100.341674   4.628771           0
## 4  9.092223  181.101509  17978.986339  ...        31.997993   4.075075           0
##
## [5 rows x 10 columns]
```

NC STATE UNIVERSITY

Classification Tree

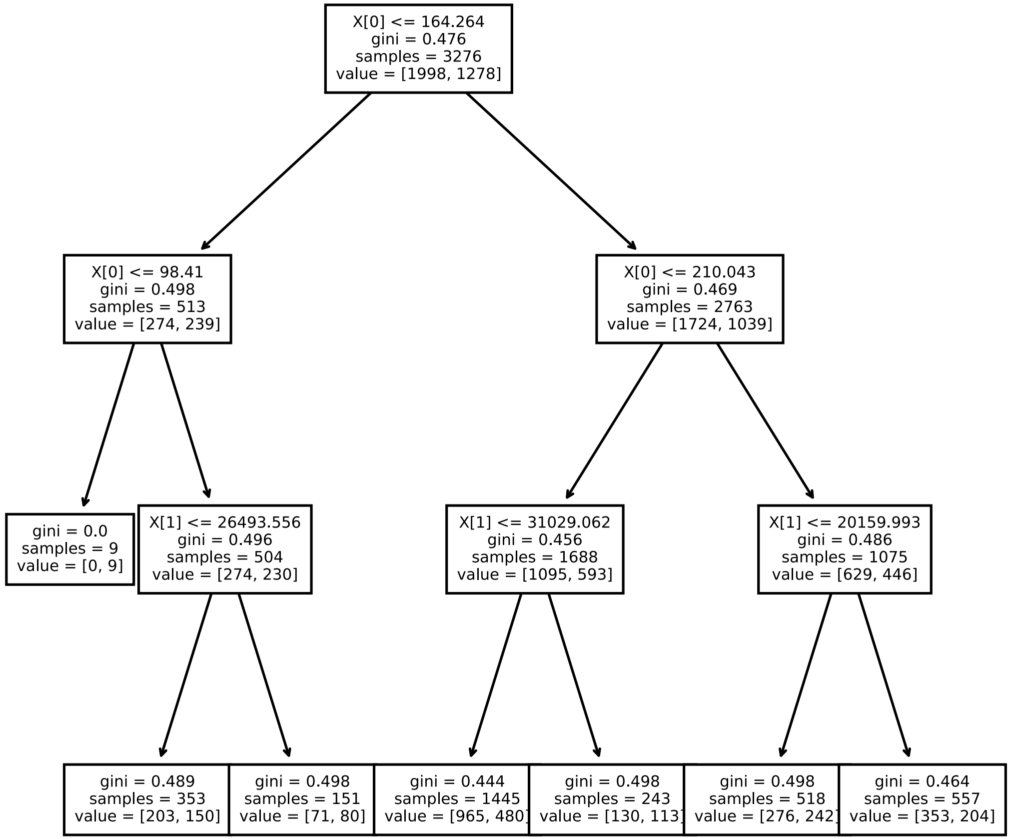
- Recall the water potability data and `Potability` as our response

```
water = pd.read_csv("data/water_potability.csv")
water.head()
```

```
##          ph    Hardness        Solids  ...  Trihalomethanes  Turbidity  Potability
## 0       NaN  204.890455  20791.318981  ...        86.990970   2.963135           0
## 1  3.716080  129.422921  18630.057858  ...        56.329076   4.500656           0
## 2  8.099124  224.236259  19909.541732  ...        66.420093   3.055934           0
## 3  8.316766  214.373394  22018.417441  ...       100.341674   4.628771           0
## 4  9.092223  181.101509  17978.986339  ...        31.997993   4.075075           0
##
## [5 rows x 10 columns]
```

- Use a similar function to fit classification trees

```
from sklearn.tree import DecisionTreeClassifier
cltree1 = DecisionTreeClassifier(max_depth=3)
cltree1.fit(water[['Hardness', 'Solids']].values, water['Potability'].values)
```


X[0] <= 164.264
gini = 0.476
samples = 3276
value = [1998, 1278]
X[0] <= 98.41
gini = 0.498
samples = 513
value = [274, 239]
X[0] <= 210.043
gini = 0.469
samples = 2763
value = [1724, 1039]
gini = 0.0
samples = 9
value = [0, 9]
X[1] <= 26493.556
gini = 0.496
samples = 504
value = [274, 230]
X[1] <= 31029.062
gini = 0.456
samples = 1688
value = [1095, 593]
X[1] <= 20159.993
gini = 0.486
samples = 1075
value = [629, 446]
gini = 0.489
samples = 353
value = [203, 150]
gini = 0.498
samples = 151
value = [71, 80]
gini = 0.444
samples = 1445
value = [965, 480]
gini = 0.498
samples = 243
value = [130, 113]
gini = 0.498
samples = 518
value = [276, 242]
gini = 0.464
samples = 557
value = [353, 204]

Predictions

- Model fit can be used for the same types of predictions as logistic regression

```
cltree1.predict(np.array([[175, 15666],
                          [217, 15666],
                          [196, 22014],
                          [217, 22014]]))
```

```
## array([0, 0, 0, 0], dtype=int64)
```

```
cltree1.predict_proba(np.array([[175, 15666],
                          [217, 15666],
                          [196, 22014],
                          [217, 22014]]))
```

```
## array([[0.66782007, 0.33217993],
##        [0.53281853, 0.46718147],
##        [0.66782007, 0.33217993],
##        [0.63375224, 0.36624776]])
```

Pruning the Tree

- Tree fit can depend on max depth and how many splits on each branch
- Often a large tree is fit and **pruned** back
    - We can also control the minimum number of samples a leaf can have via `min_samples_leaf`
    - Many other things we could consider, but let's focus on those two!

Pruning the Tree

- Tree fit can depend on max depth and how many splits on each branch
- Often a large tree is fit and **pruned** back
  - We can also control the minimum number of samples a leaf can have via `min_samples_leaf`
  - Many other things we could consider, but let's focus on those two!
- Best combination of these two can be determined using cross-validation!
  - Set up values to consider
  - Use `GridSearchCV()` to return the best values

Pruning the Tree

- Set up our values to consider as a dictionary

```
parameters = {'max_depth': range(2,15),
              'min_samples_leaf':[3, 5, 10, 50, 100]}
```

NC STATE UNIVERSITY

Pruning the Tree

- Set up our values to consider as a dictionary

```
parameters = {'max_depth': range(2,15),
              'min_samples_leaf':[3, 5, 10, 50, 100]}
```

- Import the grid search function

```
from sklearn.model_selection import GridSearchCV
tuning_model = GridSearchCV(DecisionTreeClassifier(),
                            parameters,
                            cv = 5,
                            scoring='accuracy')
```

Pruning the Tree

- Now we fit the model

```
tuning_model.fit(water[['Hardness', 'Solids', 'Chloramines', 'Organic_carbon']].values,
                 water['Potability'].values)
```

NC STATE UNIVERSITY

Pruning the Tree

- Now we fit the model

```
tuning_model.fit(water[['Hardness', 'Solids', 'Chloramines', 'Organic_carbon']].values,
                 water['Potability'].values)
```

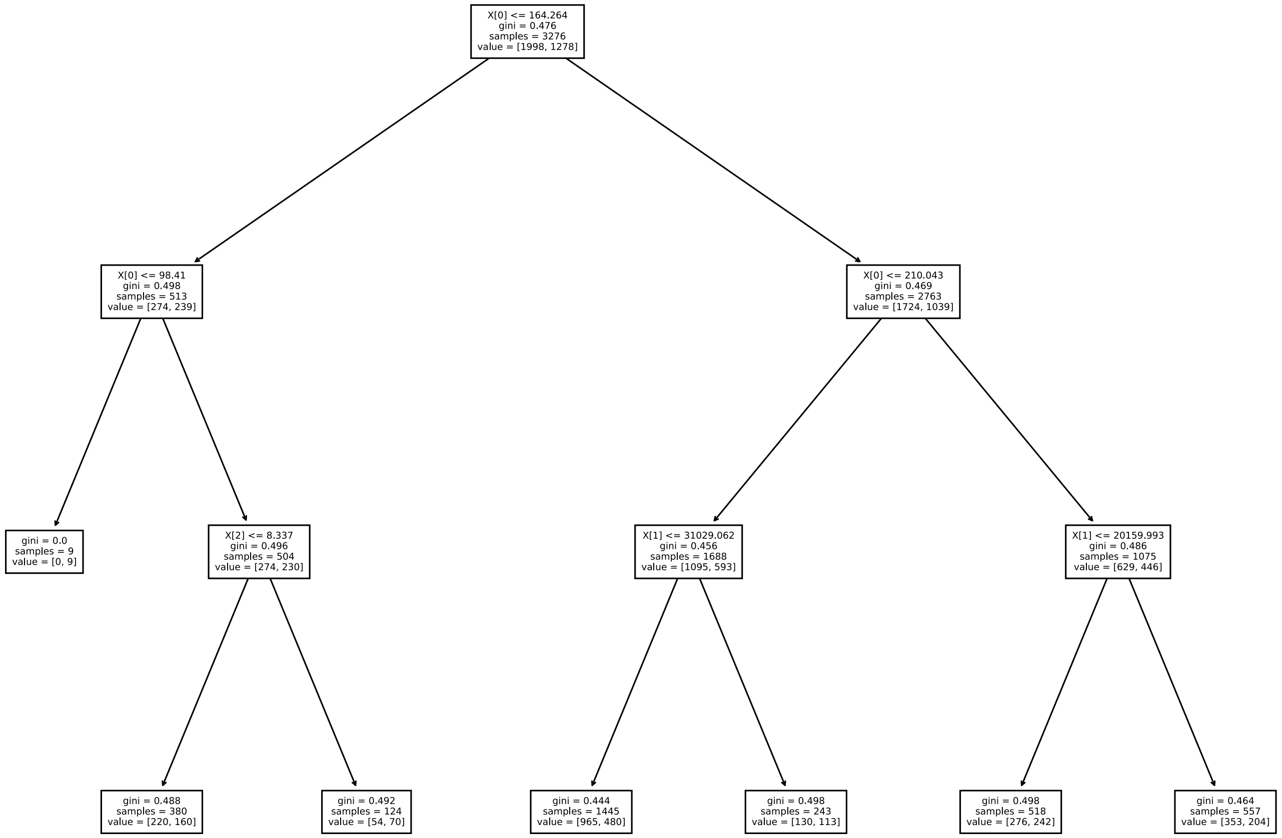
- Inspect the best tuning parameters

```
print(tuning_model.best_estimator_)
```

```
## DecisionTreeClassifier(max_depth=3, min_samples_leaf=3)
```

```
print(tuning_model.best_score_, tuning_model.best_params_)
```

```
## 0.6098896853472352 {'max_depth': 3, 'min_samples_leaf': 3}
```


X[0] <= 164.264
gini = 0.476
samples = 3276
value = [1998, 1278]
X[0] <= 98.41
gini = 0.498
samples = 513
value = [274, 239]
X[0] <= 210.043
gini = 0.469
samples = 2763
value = [1724, 1039]
gini = 0.0
samples = 9
value = [0, 9]
X[2] <= 8.337
gini = 0.496
samples = 504
value = [274, 230]
X[1] <= 31029.062
gini = 0.456
samples = 1688
value = [1095, 593]
X[1] <= 20159.993
gini = 0.486
samples = 1075
value = [629, 446]
gini = 0.488
samples = 380
value = [220, 160]
gini = 0.492
samples = 124
value = [54, 70]
gini = 0.444
samples = 1445
value = [965, 480]
gini = 0.498
samples = 243
value = [130, 113]
gini = 0.498
samples = 518
value = [276, 242]
gini = 0.464
samples = 557
value = [353, 204]

Different Metric

- Instead of misclassification, consider `neg_log_loss`

```
tuning_model2 = GridSearchCV(DecisionTreeClassifier(),
                            parameters,
                            cv = 5,
                            scoring='neg_log_loss')
tuning_model2.fit(water[['Hardness', 'Solids', 'Chloramines', 'Organic_carbon', 'Turbidity']].values,
                  water['Potability'].values)
```

```
print(tuning_model2.best_estimator_)
```

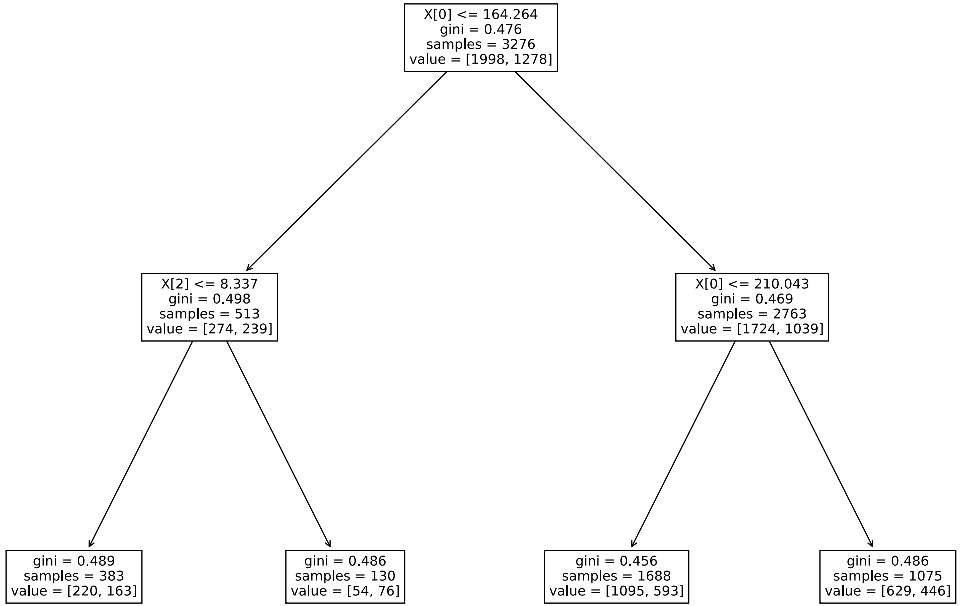
```
## DecisionTreeClassifier(max_depth=2, min_samples_leaf=100)
```

```
print(tuning_model2.best_score_, tuning_model2.best_params_)
```

```
## -0.6743150518917174 {'max_depth': 2, 'min_samples_leaf': 100}
```

```
tuning_model2.predict_proba(np.array([[150, 0, 8.5, 0, 0]]))
```

```
## array([[0.41538462, 0.58461538]])
```



Recap

- Trees are a nonlinear model

Pros:

- Simple to understand and easy to interpret output
- Predictors don't need to be scaled
- No statistical assumptions necessary
- Built in variable selection

Cons:

- Small changes in data can vastly change tree
- No optimal algorithm for choosing splits
- Need to prune

Bagging Trees, Random Forests, and Boosted Trees are three methods that average across trees. Lose interpretability but gain in prediction!