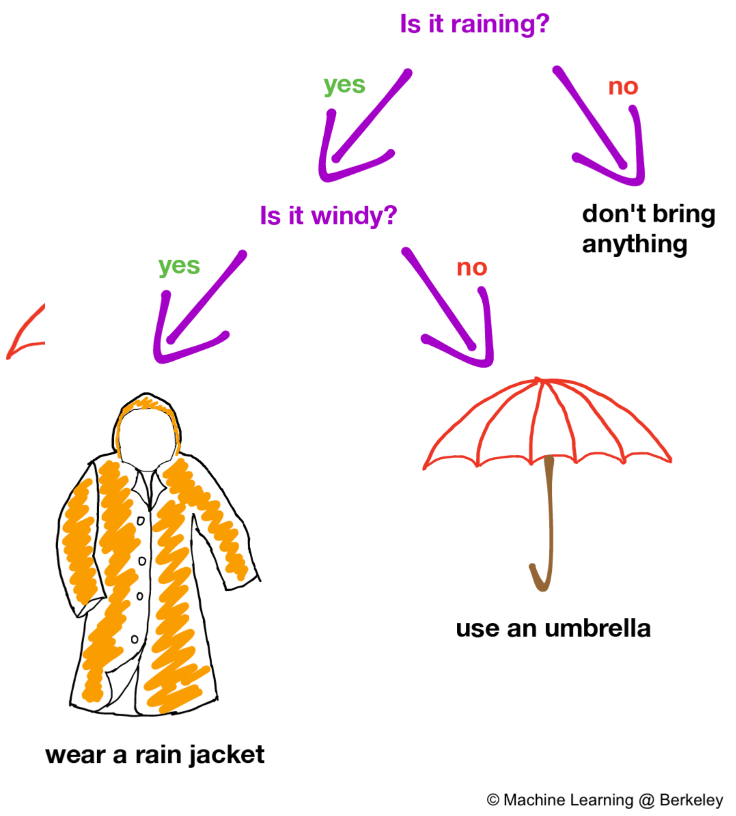
Applied Data Science

L11. Decision Trees

Irina Mohorianu

Head of Bioinformatics/ Scientific Computing @CSCI

Decision trees. Definitions

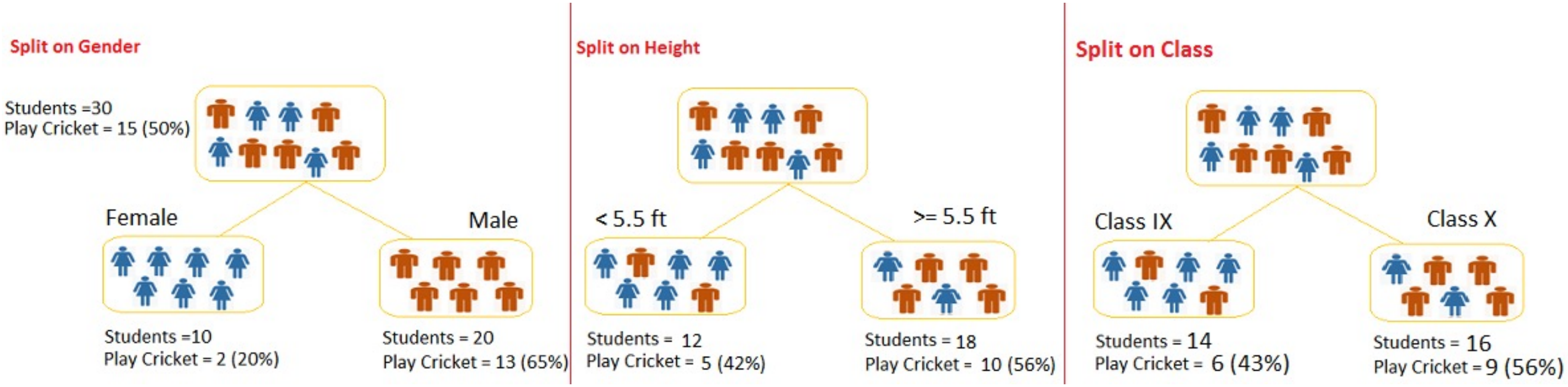


What is a Decision Tree?

Decision tree or recursive partitioning is a supervised graph based algorithm to represent choices and the results of the choices in the form of a tree.

The nodes in the graph represent an event or choice and it is referred to as a leaf and the set of decisions made at the node is referred to as branches.

Decision trees map non-linear relationships and the hierarchical leaves and branches make a Tree.

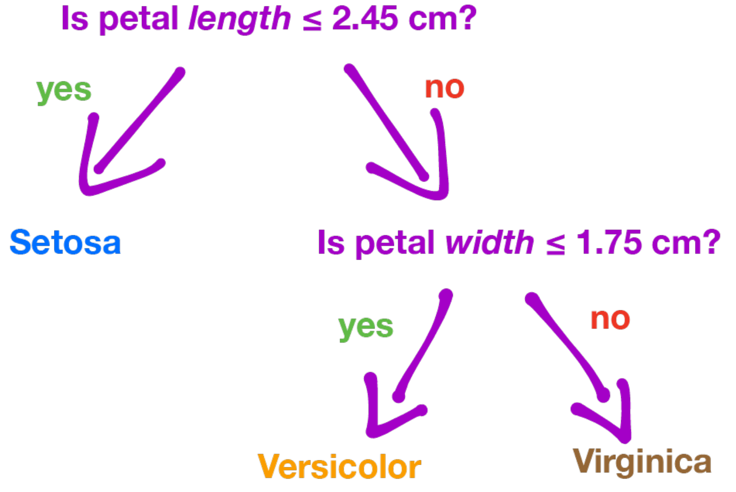


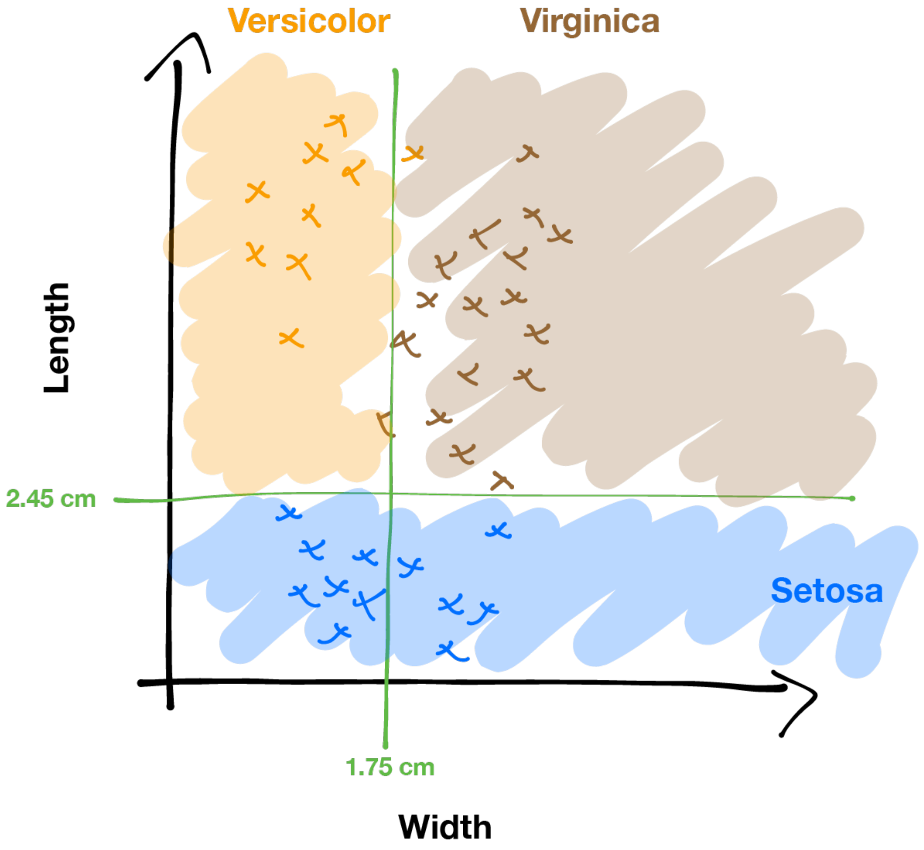
Decision trees. Definitions
Splitting decision

The classification tree searches through each dependent variable to find a single variable that splits the data into two or more groups and this process is repeated until the stopping criteria is invoked.

The decision of making strategic splits heavily affects a tree's accuracy. The decision criteria is different for classification and regression trees.

Decision trees use multiple algorithms to decide to split a node in two or more sub-nodes. The common goal for these algorithms is the creation of sub-nodes with increased homogeneity. In other words, we can say that purity of the node increases with respect to the target variable. Decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

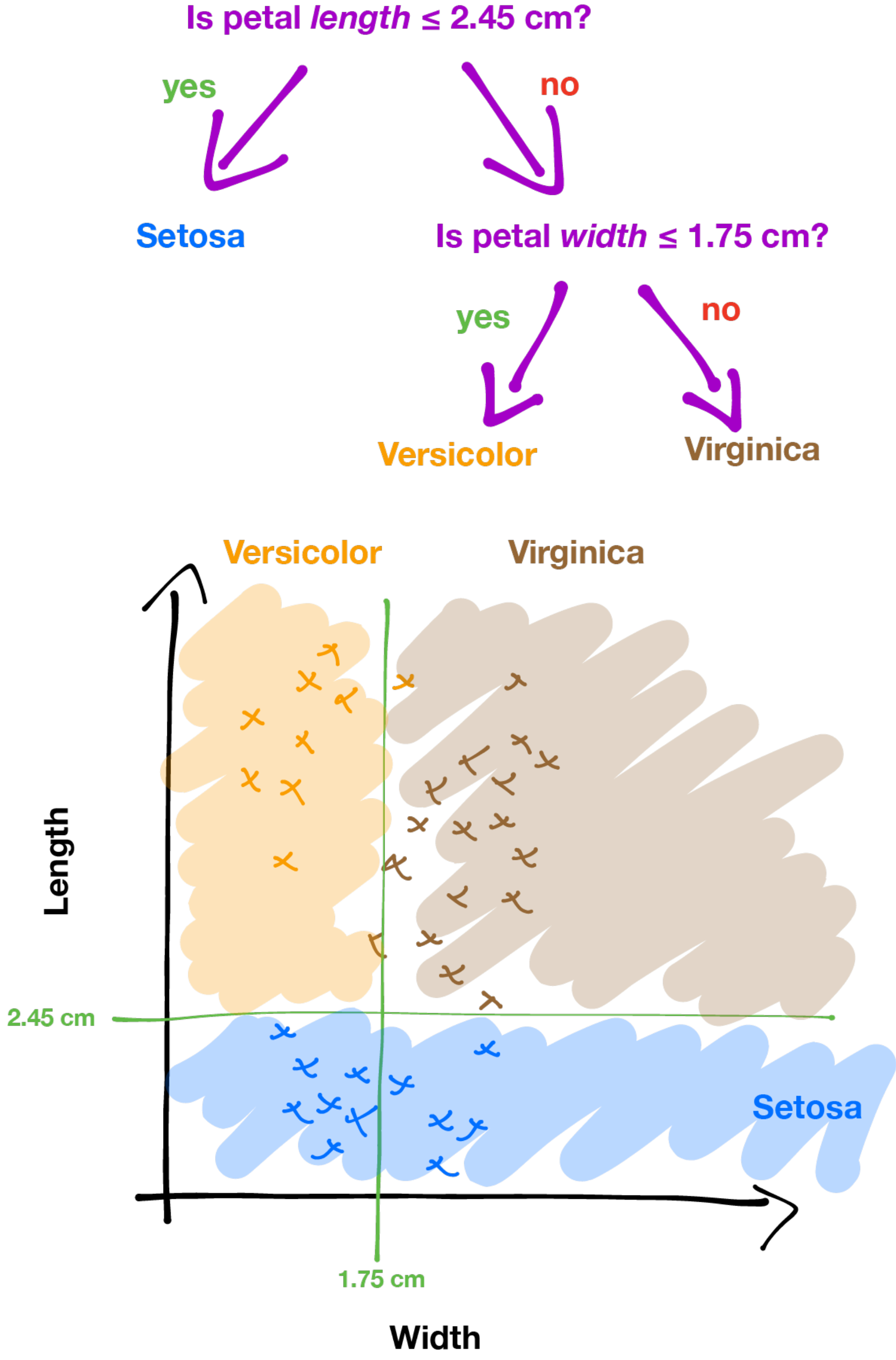




Decision trees
Splitting criteria

Node splitting, or simply splitting, divides a node into multiple sub-nodes to create relatively pure nodes. This is done by finding the best split for a node and can be done in multiple ways. The splitting can be broadly divided into two categories based on the type of target variable:

1. **Continuous Target Variable:** Reduction in Variance
2. **Categorical Target Variable:**
   Gini Impurity, Information Gain, and Chi-Square



Decision trees
Splitting criteria

Gini Index

The Gini Index or Gini Impurity is calculated by subtracting the sum of the squared probabilities of each class from one. It favours mostly the larger partitions and are very simple to implement. In simple terms, it calculates the probability of a certain randomly selected feature that was classified incorrectly.

The Gini Index varies between 0 and 1, where 0 represents purity of the classification and 1 denotes random distribution of elements among various classes. A Gini Index of 0.5 shows that there is equal distribution of elements across some classes.

$$G = \sum_{i=1}^{C} p(i) * (1 - p(i))$$

The Gini Index works on categorical variables and gives the results in terms of “success” or “failure” and hence performs only binary split. It isn’t computationally intensive as its counterpart – Information Gain. From the Gini Index, the value of another parameter named Gini Gain is calculated whose value is maximised with each iteration by the Decision Tree to get the perfect CART

Decision trees
Splitting criteria

Information Gain

$$Information\,Gain = 1 - Entropy$$

Entropy is used for calculating the purity of a node. The lower the value of entropy, the higher the purity of the node. The entropy of a homogeneous node is zero. Since we subtract entropy from 1, the Information Gain is higher for the purer nodes with a maximum value of 1.

$$Entropy = -\sum_{i=1}^{n} p_i \log_2 p_i$$

Steps to split a decision tree using Information Gain:

[1] For each split, individually calculate the entropy of each child node
[2] Calculate the entropy of each split as the weighted average entropy of child nodes
[3] Select the split with the lowest entropy or highest information gain
[4] Until you achieve homogeneous nodes, repeat steps 1-3

Decision trees. Splitting criteria

ChiSq approach – multiple output classes

$$Chi\text{-}Square = \sqrt{\frac{(Actual - Expected)^2}{Expected}}$$

the Expected is the expected value for a class in a child node based on the distribution of classes in the parent node, and the Actual is the actual value for a class in a child node.

The higher the value, the higher will be the differences between parent and child nodes, i.e., the higher will be the homogeneity.

[1] For each split, individually calculate the Chi-Sq value of each child node

i.e. taking the sum of Chi-Sq values for each class in a node

[2] Calculate the Chi-Sq value of each split as the sum of Chi-Sq values for all the child nodes

[3] Select the split with a higher Chi-Square value

[4] Until you achieve homogeneous nodes, repeat steps 1-3

Decision trees
Splitting criteria. Continuous output

Variance is used for calculating the homogeneity of a node. If a node is entirely homogeneous, then the variance is zero.

The steps to split a node in a tree using the reduction in variance method:

[1] For each split, individually calculate the variance of each child node
[2] Calculate the variance of each split as the weighted average variance of child nodes
[3] Select the split with the lowest variance
[4] Perform steps 1-3 until completely homogeneous nodes are achieved

$$Variance = \frac{\sum (X - \mu)^2}{N}$$

Decision trees
Advantages. Disadvantages

Some parameters used for building a tree and constrain overfitting

Minimum sample for a node split
Minimum sample for a terminal node
Maximum depth of a tree
Maximum number of terminal nodes
Maximum features considered for a split

Advantages of decision tree

[1] Simple to understand and use
[2] Algorithms are robust to noisy data
[3] Useful in data exploration
decision tree is non parametric no assumptions on the distribution of variables

Disadvantages of decision tree

1.Overfitting is the common disadvantage of decision trees [also trees can get trapped into local optima]. Can be partially addressed by constraining the model parameter and by pruning.
2. It is not ideal for continuous variables as in it looses information

Decision trees.
Example 1. Car dataset

```
car_data = pd.read_csv("car_data.csv", header=None)
car_data = car_data.iloc[1:, 1:]
car_data.columns = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']

print(car_data.shape)
print(car_data.head())
```

```
(1728, 7)
  buying  maint doors persons lug_boot safety  class
1  vhigh  vhigh     2       2    small    low  unacc
2  vhigh  vhigh     2       2    small    med  unacc
3  vhigh  vhigh     2       2    small   high  unacc
4  vhigh  vhigh     2       2      med    low  unacc
5  vhigh  vhigh     2       2      med    med  unacc
```

```
# Splitting the data
X = car_data.drop(columns=[car_data.columns[-1]])
y = car_data[car_data.columns[-1]]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print(X_train.shape, X_test.shape)
```

```
(1209, 6) (519, 6)
```

Decision trees
Example 1. Car dataset

```
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=10, random_state=42)
```

```
# Columns for ordinal encoding
ordinal_cols = ['persons', 'doors']

# Columns for one-hot encoding
one_hot_cols = ['buying', 'lug_boot', 'maint', 'safety']

# Define the encoding transformation
transformers = [
    ('ordinal', OrdinalEncoder(categories=[['2', '4', 'more'], ['2', '3', '4', '5more']]), ordinal_cols),
    ('onehot', OneHotEncoder(drop='first', sparse=False), one_hot_cols)
]

preprocessor = ColumnTransformer(transformers, remainder='passthrough')
```

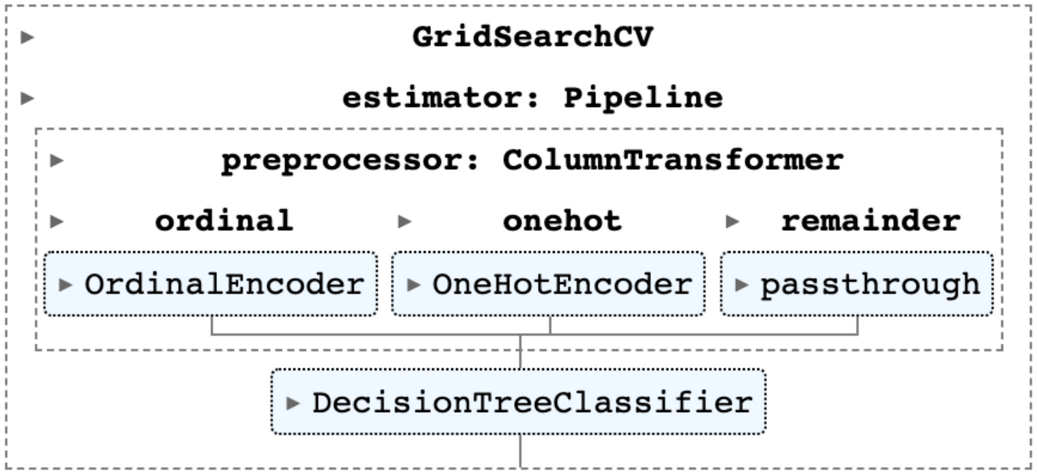
```
# Create pipelines
pipeline_gini = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', DecisionTreeClassifier(criterion='gini'))
])

pipeline_entropy = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', DecisionTreeClassifier(criterion='entropy'))
])

# Set up GridSearchCV for both pipelines
```

Decision trees
Example 1. Car dataset

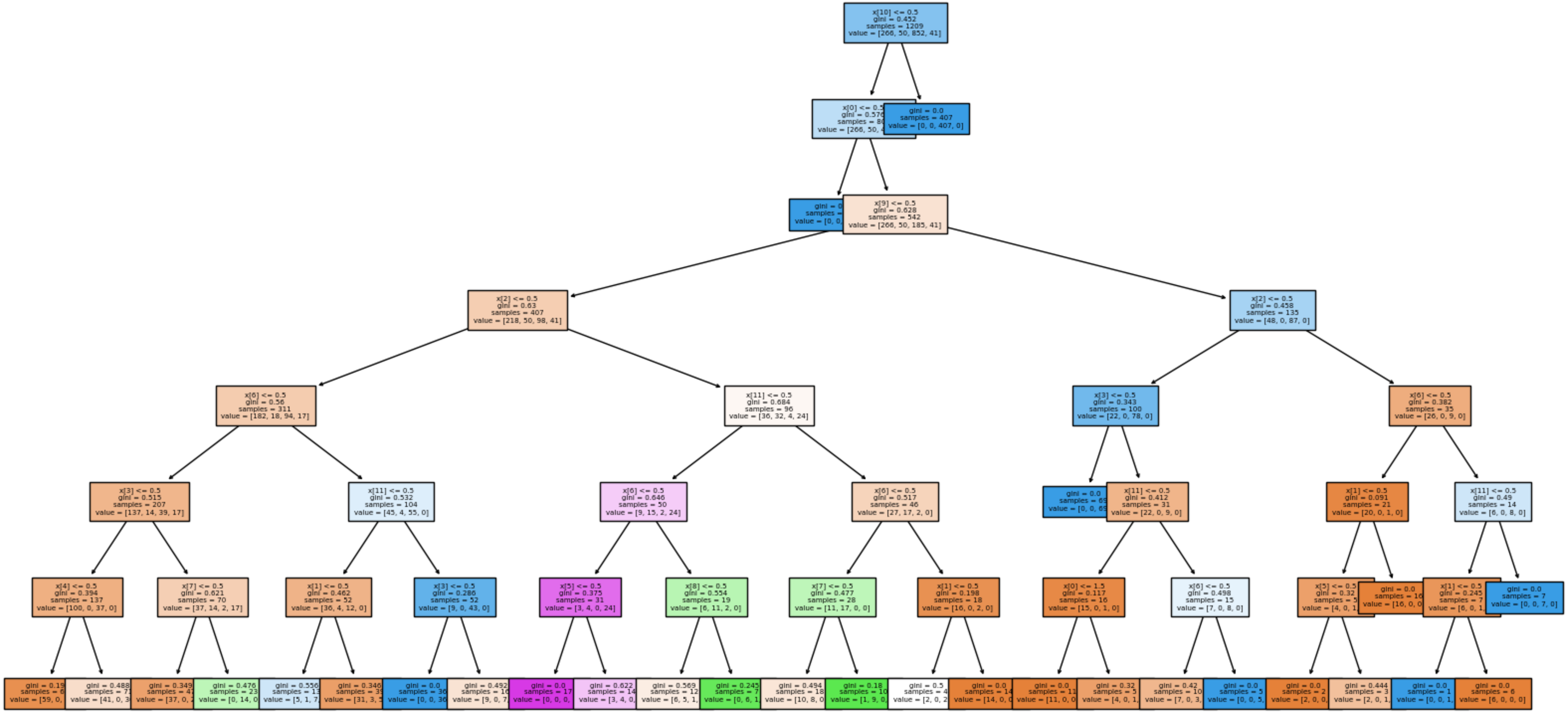
```python
# Create pipelines
pipeline_gini = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', DecisionTreeClassifier(criterion='gini'))
])

pipeline_entropy = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', DecisionTreeClassifier(criterion='entropy'))
])

# Set up GridSearchCV for both pipelines
param_grid = {'classifier__max_depth': range(1, 8)}

grid_gini = GridSearchCV(pipeline_gini, param_grid=param_grid, cv=cv)
grid_entropy = GridSearchCV(pipeline_entropy, param_grid=param_grid, cv=cv)

grid_gini.fit(X_train, y_train)
print("Best parameters (Gini):", grid_gini.best_params_)

grid_entropy.fit(X_train, y_train)
print("Best parameters (Entropy):", grid_entropy.best_params_)
```

```
Best parameters (Gini): {'classifier__max_depth': 7}
Best parameters (Entropy): {'classifier__max_depth': 7}
```

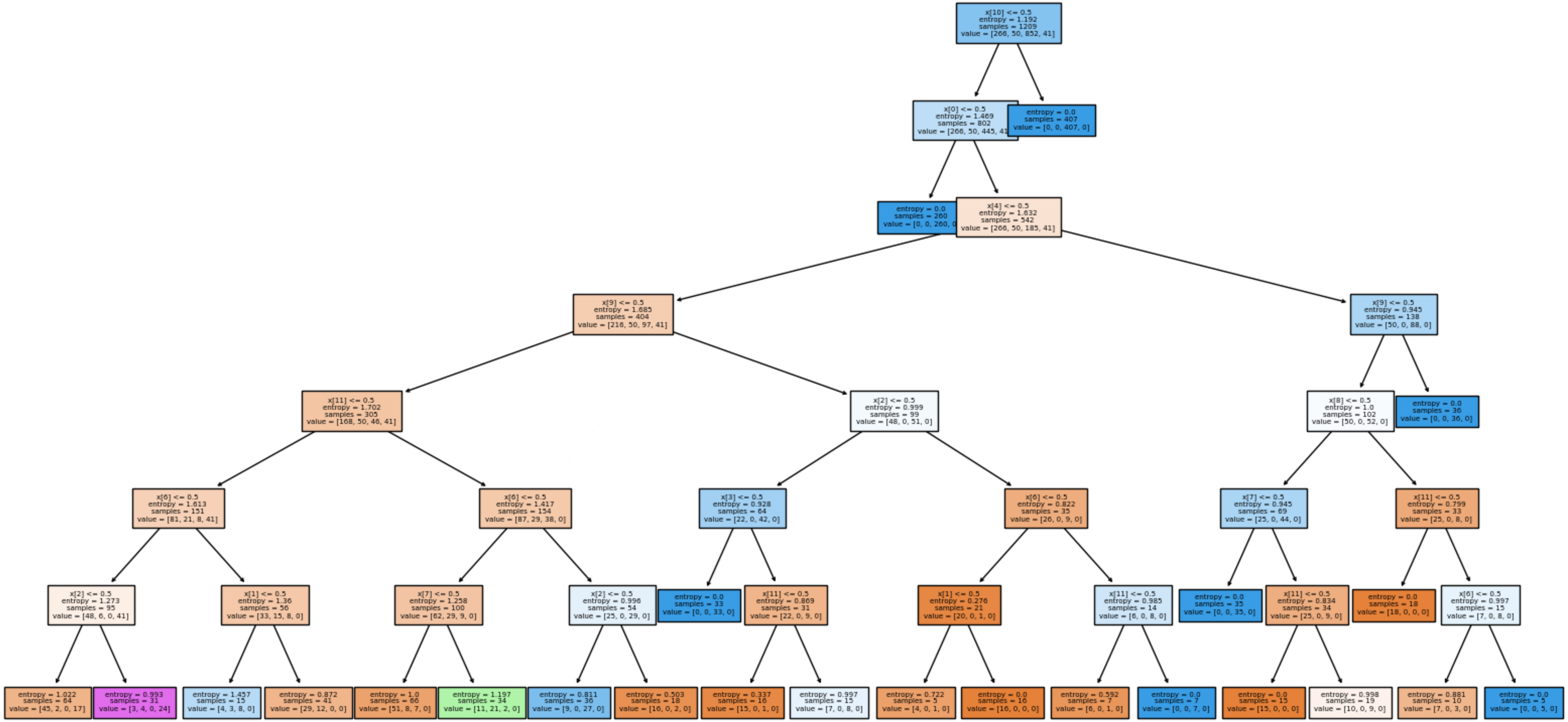
```python
grid_gini
```



Decision trees
Example 1. Car dataset. Gini

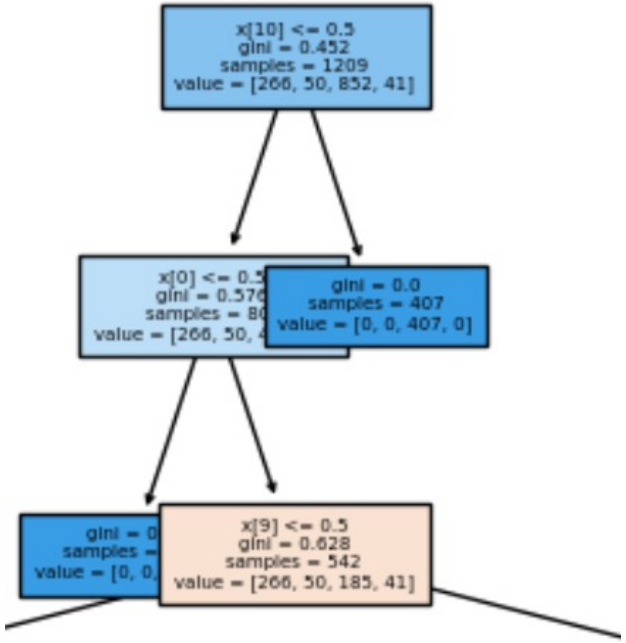


Decision trees
Example 1. Car dataset. Entropy

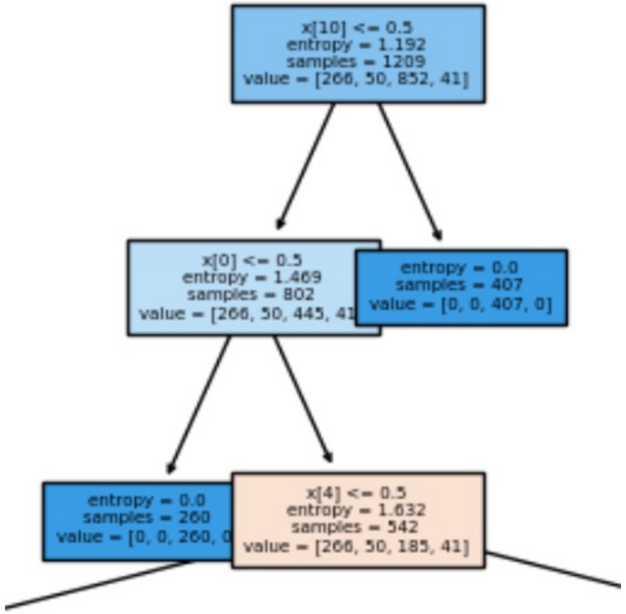


Decision trees
Example 1. Car dataset

Gini index

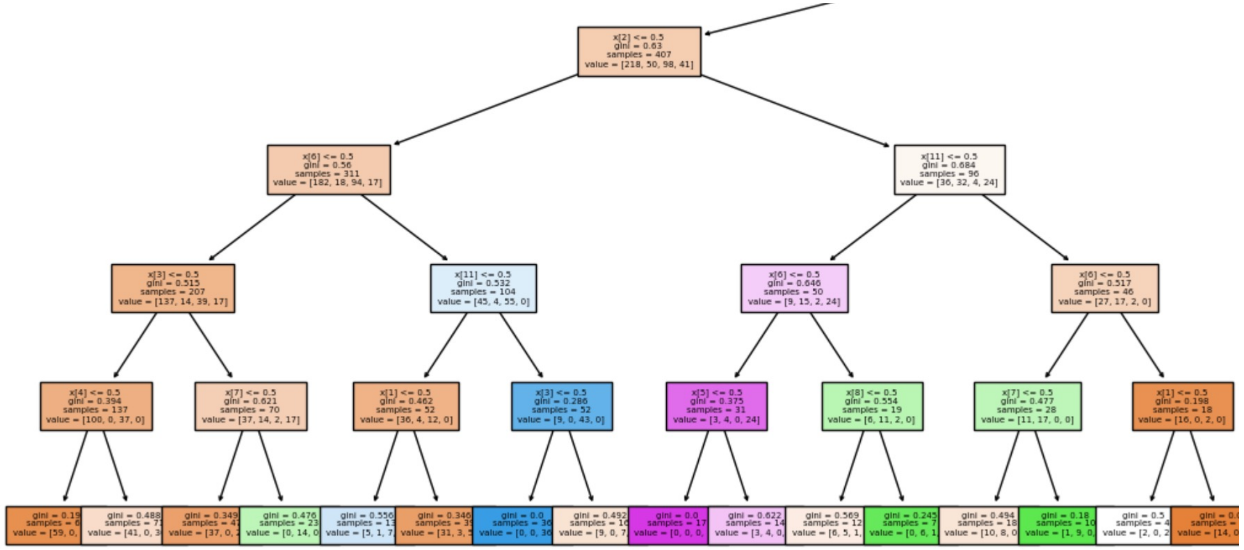


Entropy

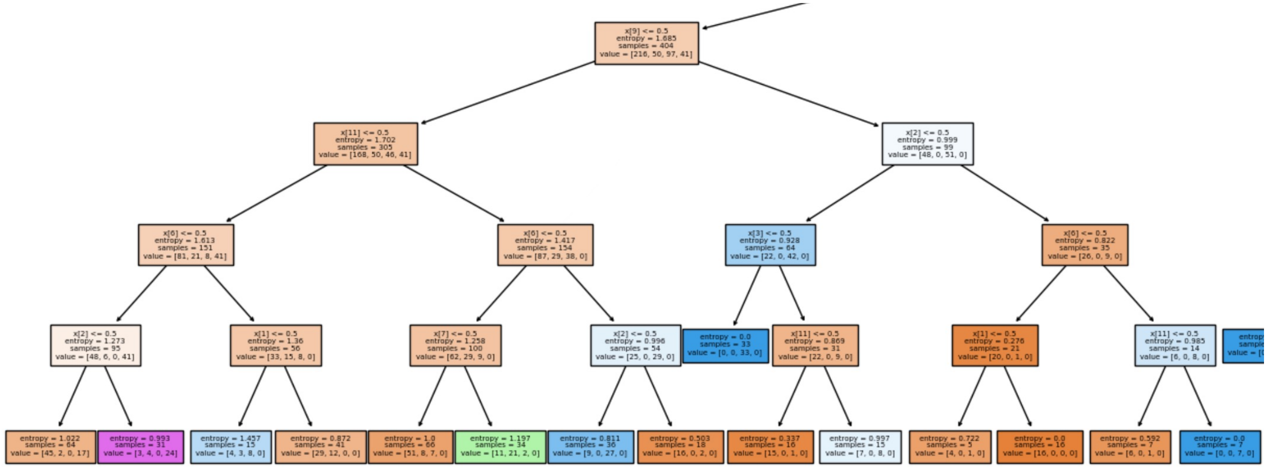


Decision trees
Example 1. Car dataset

Gini index

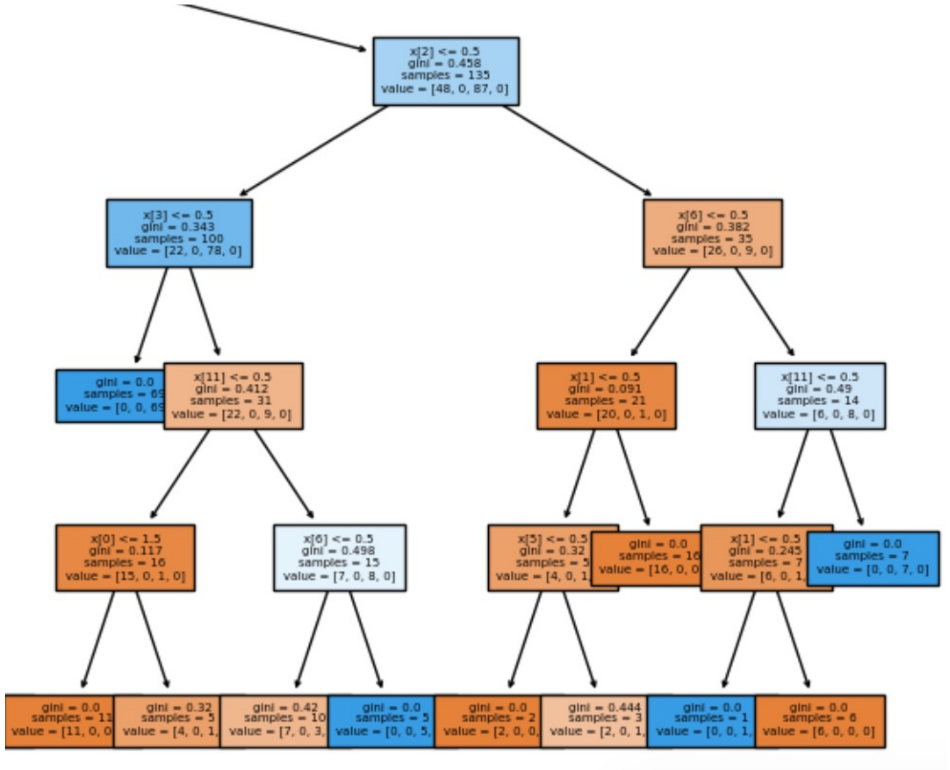


Entropy

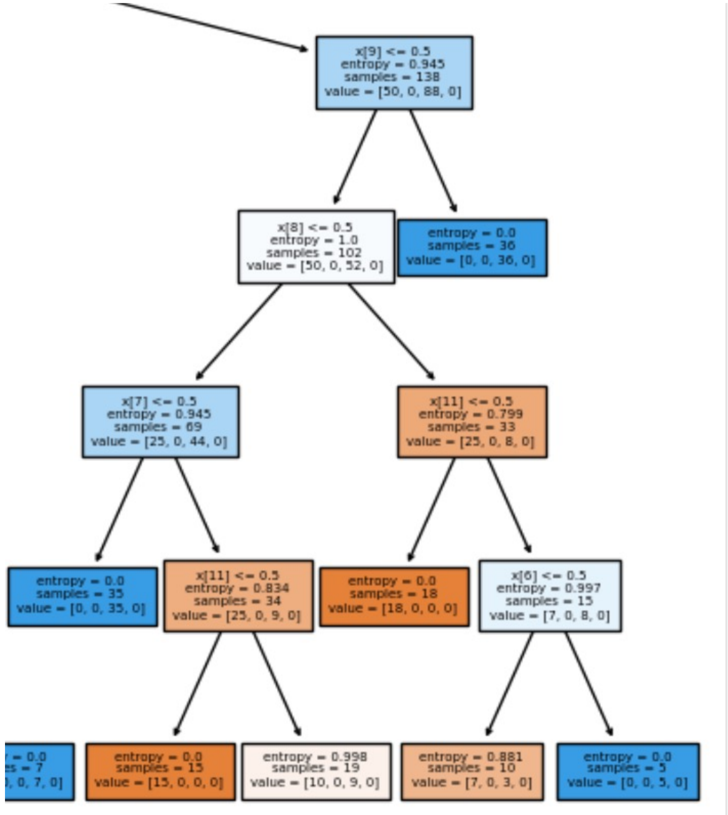


Decision trees
Example 1. Car dataset

Gini index



Entropy



Decision trees
Example 1. Car dataset

```python
# Prediction and evaluation for Gini
y_pred_train_gini = grid_gini.predict(X_train)
y_pred_test_gini = grid_gini.predict(X_test)

print("Prediction and evaluation for Gini model")
print("Train")
print(confusion_matrix(y_train, y_pred_train_gini))

print("Test")
print(confusion_matrix(y_test, y_pred_test_gini))
print(classification_report(y_test, y_pred_test_gini))
```

```
Prediction and evaluation for Gini model
Train
[[35  0  0]
 [ 0 34  1]
 [ 0  1 34]]
Test
[[15  0  0]
 [ 0 15  0]
 [ 0  1 14]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        15
         1.0       0.94      1.00      0.97        15
         2.0       1.00      0.93      0.97        15

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

```python
# Prediction and evaluation for Entropy
y_pred_train_entropy = grid_entropy.predict(X_train)
y_pred_test_entropy = grid_entropy.predict(X_test)

print("Prediction and evaluation for entropy model")
print("Train")
print(confusion_matrix(y_train, y_pred_train_entropy))

print("Test")
print(confusion_matrix(y_test, y_pred_test_entropy))
print(classification_report(y_test, y_pred_test_entropy
```

```
Prediction and evaluation for entropy model
Train
[[35  0  0]
 [ 0 33  2]
 [ 0  0 35]]
Test
[[15  0  0]
 [ 0 12  3]
 [ 0  0 15]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        15
         1.0       1.00      0.80      0.89        15
         2.0       0.83      1.00      0.91        15

    accuracy                           0.93        45
   macro avg       0.94      0.93      0.93        45
weighted avg       0.94      0.93      0.93        45
```

Decision trees
Cell Segmentation example

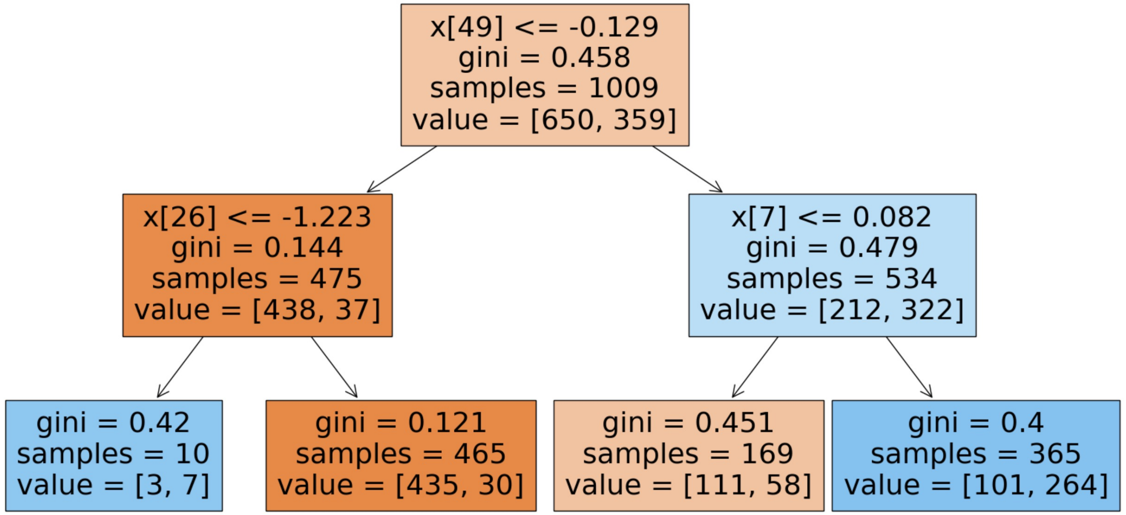
```
data = pd.read_csv('segmentation_data.csv')
data = data.drop("Unnamed: 0", axis=1)

X = data.drop(columns=['Class'])
X = X.iloc[:,2:]
y = data['Class']
```

```
# Partition data
X_train, X_test, y_train, y_test = t
```

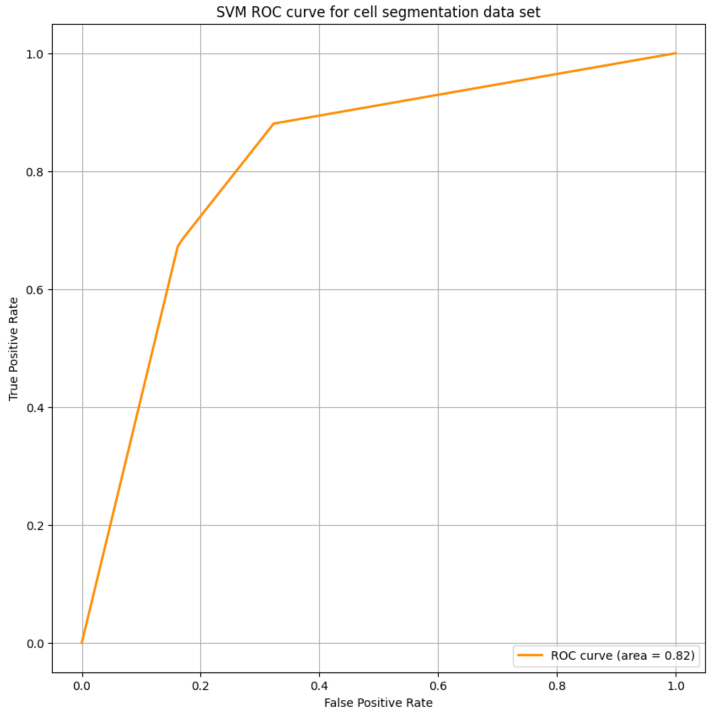
```
# Cross-validation
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=5, random_state=42)
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_auc_score, roc_curve, confusion_matrix

preprocessor = Pipeline([
    ('scaler', StandardScaler())
])

dtree = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', DecisionTreeClassifier())
])

param_grid = {
    'classifier__max_depth': [None, 2, 4, 6, 8, 10, 12]
}

grid_search_dtree = GridSearchCV(dtree, param_grid, cv=cv, verbose=1)
grid_search_dtree.fit(X_train, y_train)
```

Decision trees
Cell Segmentation example




```
[[539 111]
 [113 247]]
```



Area under the curve: 0.816367521

Decision trees
Cell Segmentation example

```
# Testing set predictions
y_pred_dt = grid_search_dtree.predict(X_test)
print(confusion_matrix(y_test, y_pred_dt))
print(classification_report(y_test, y_pred_dt))
```

```
[[539 111]
 [113 247]]
              precision    recall  f1-score   support

          PS       0.83      0.83      0.83       650
          WS       0.69      0.69      0.69       360

    accuracy                           0.78      1010
   macro avg       0.76      0.76      0.76      1010
weighted avg       0.78      0.78      0.78      1010
```

Decision trees
Blood brain barrier example

```
from sklearn.tree import DecisionTreeRegressor
```

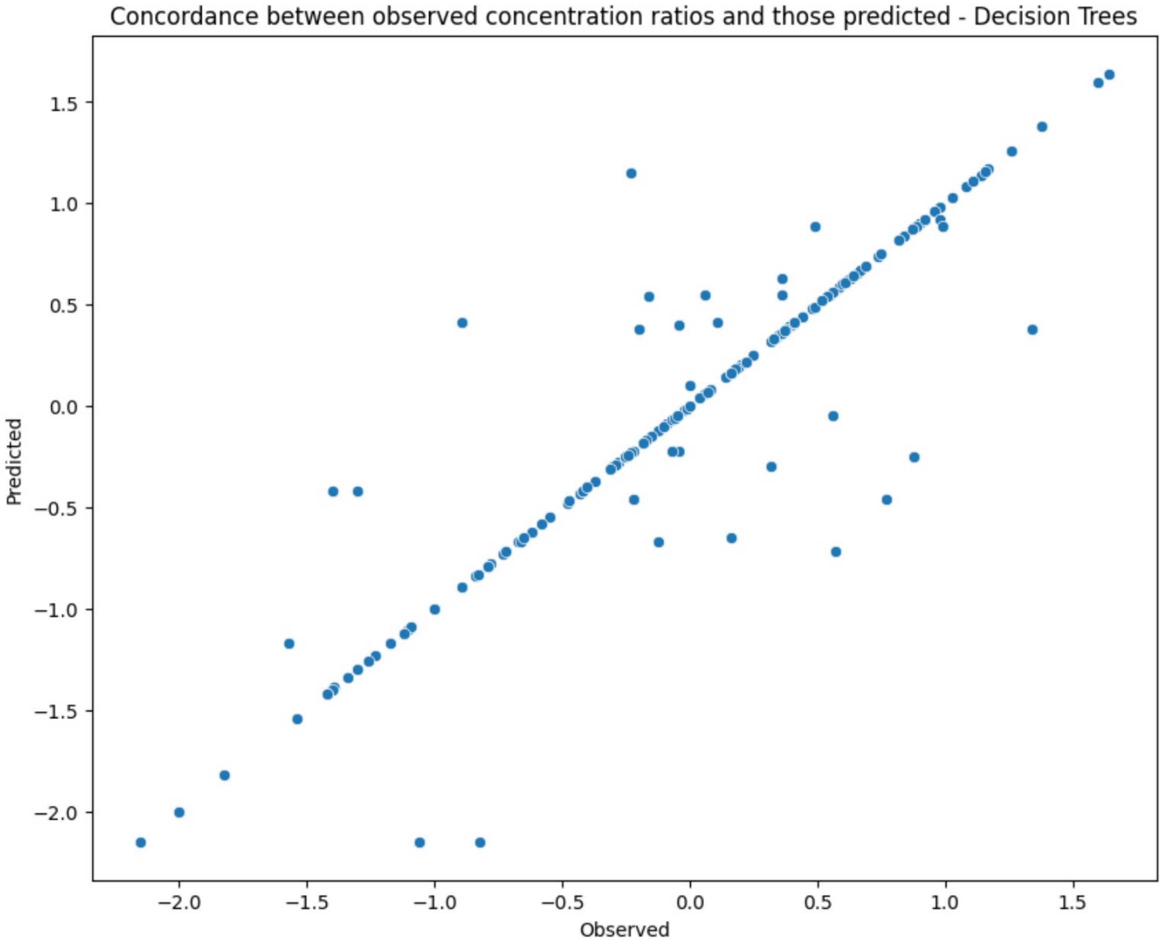
```
data = pd.read_csv('bbb_df.csv')
data = data.drop("Unnamed: 0", axis=1)

descr = data.drop(columns='logBBB')
logBBB = data['logBBB']

train_index, test_index = train_test_split(descr.index, test_size=0.2, random_state=42,)
descr_train = descr.iloc[train_index]
conc_ratio_train = logBBB.iloc[train_index]
descr_test = descr.iloc[-train_index]
conc_ratio_test = logBBB.iloc[-train_index]
```

Decision trees
Blood brain barrier example

```python
# Preprocessing
numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ('boxcox', PowerTransformer(method='yeo-johnson'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, descr.columns)
])
```

```python
from sklearn.model_selection import RepeatedKFold

dtree = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', DecisionTreeRegressor())
])

param_grid = {
    'regressor__max_depth': [None, 2, 4, 6, 8, 10, 12]
}

# 5 fold - repeats 3 times - for computational purposes.
cv = RepeatedKFold(n_splits=5, n_repeats=3, random_state=42)

grid_search_dtree = GridSearchCV(dtree, param_grid, cv=cv, verbose=1)
grid_search_dtree.fit(descr_train, conc_ratio_train)
```

Decision trees
Blood brain barrier example




```
Correlation between observed and
predicted values (Decision Tree):
0.910
```