Machine Learning and Algorithms
(Session 5)

Yi Zhang

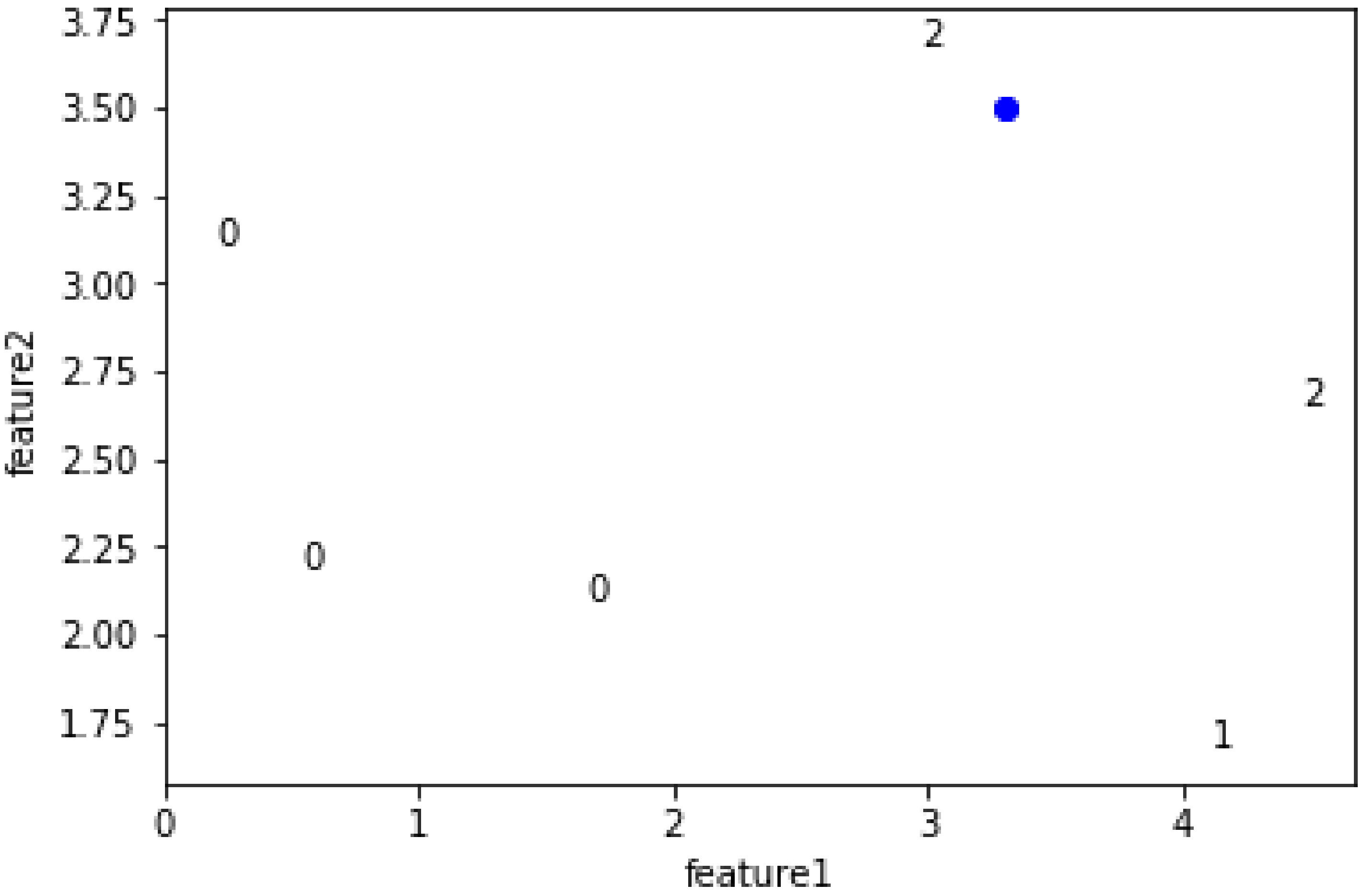
March 3, 2022

Question to solve

- Each observation has two features and a label.
- The label can be 0, 1, or 2
- For a point with x1=3.3, x2=3.5, what should be predicted label?

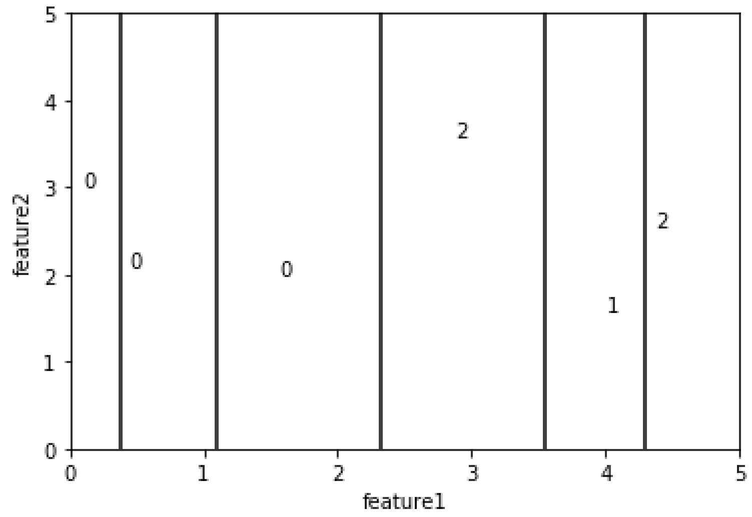
x1	x2	label
4.464301	2.649087	2
1.659899	2.094037	0
4.106146	1.677039	1
0.208483	3.112597	0
0.538283	2.190707	0
2.975260	3.679411	2

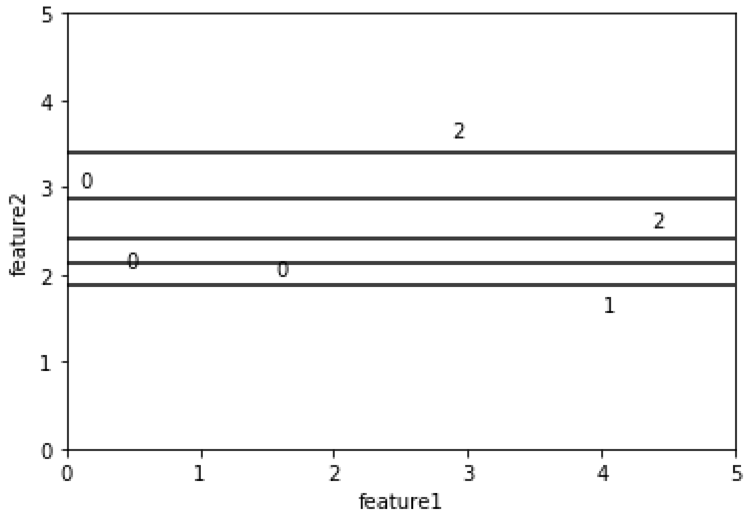
Visualize the data



Decision Tree

- Try to split the domain into segments
  - Loop through each feature
    - For each feature, loop through each mid-point between values to get the split performance
  - Choose feature that gives the best performance
  - For the final model, we aim to put observations with the same labels in one segment



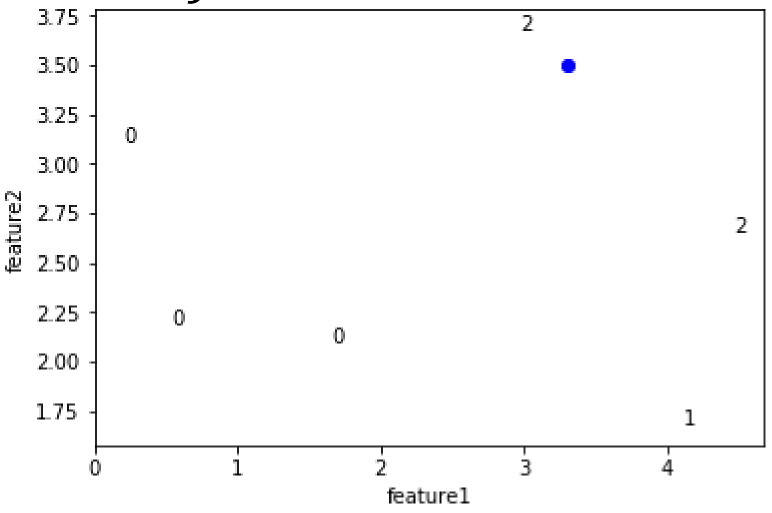


How do we decide "good" split?

- Gini Index
  - $1 - \sum_j p_j^2$, where p_j is the percentage of samples with j label
- For the root (No split at all)

$$Gini = 1 - \left(\frac{1}{6}\right)^2 - \left(\frac{3}{6}\right)^2 - \left(\frac{2}{6}\right)^2 = 1 - \frac{14}{36} = \frac{11}{18} \approx 0.611$$

  - What should be the prediction if no split?
  - Majority vote: We predict everyone as 0



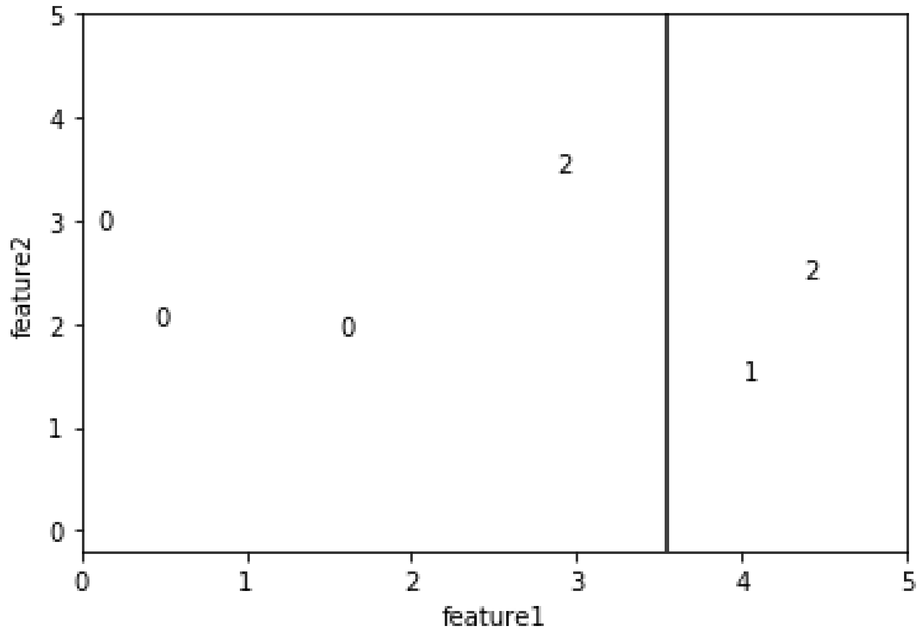
gini = 0.611
samples = 6
value = [3, 1, 2]

Splitting based-on feature 1 (1)

- Gini Index
  - $1 - \sum_j p_j^2$, where p_j is the percentage of samples with j label

- $Gini(Left) = 1 - \left(\frac{3}{4}\right)^2 - \left(\frac{1}{4}\right)^2 = \frac{3}{8} = 0.375$
- $Gini(Right) = 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = \frac{1}{2} = 0.5$

- $Gini(Weighted) = \frac{\#Left}{\#Total} Gini(Left) + \frac{\#Right}{\#Total} Gini(Right)$
$$= \frac{4}{6} * \frac{3}{8} + \frac{2}{6} * \frac{1}{2} = \frac{5}{12} \approx 0.417$$

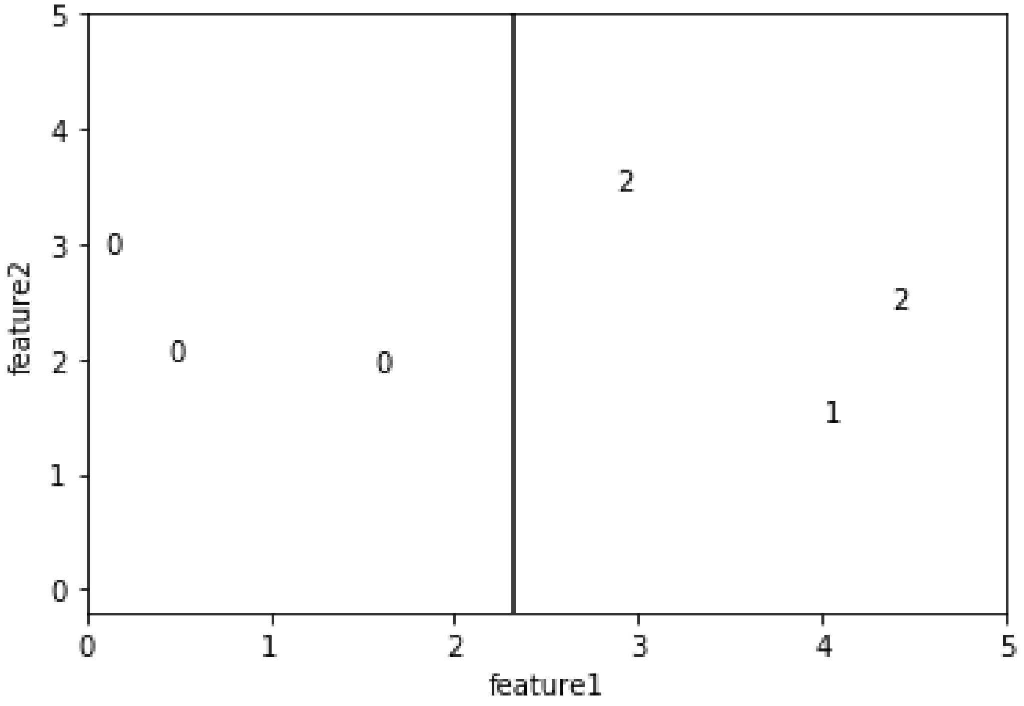


Splitting based-on feature 2 (1)

- Gini Index
  - $1 - \sum_j p_j^2$, where p_j is the percentage of samples with j label

- $Gini(Left) = 1 - 1^2 = 0$
- $Gini(Right) = 1 - \left(\frac{2}{3}\right)^2 - \left(\frac{1}{3}\right)^2 = \frac{4}{9}$ ≈0.444
- $Gini(Weighted) = \frac{\#Left}{\#Total} Gini(Left) + \frac{\#Right}{\#Total} Gini(Right)$

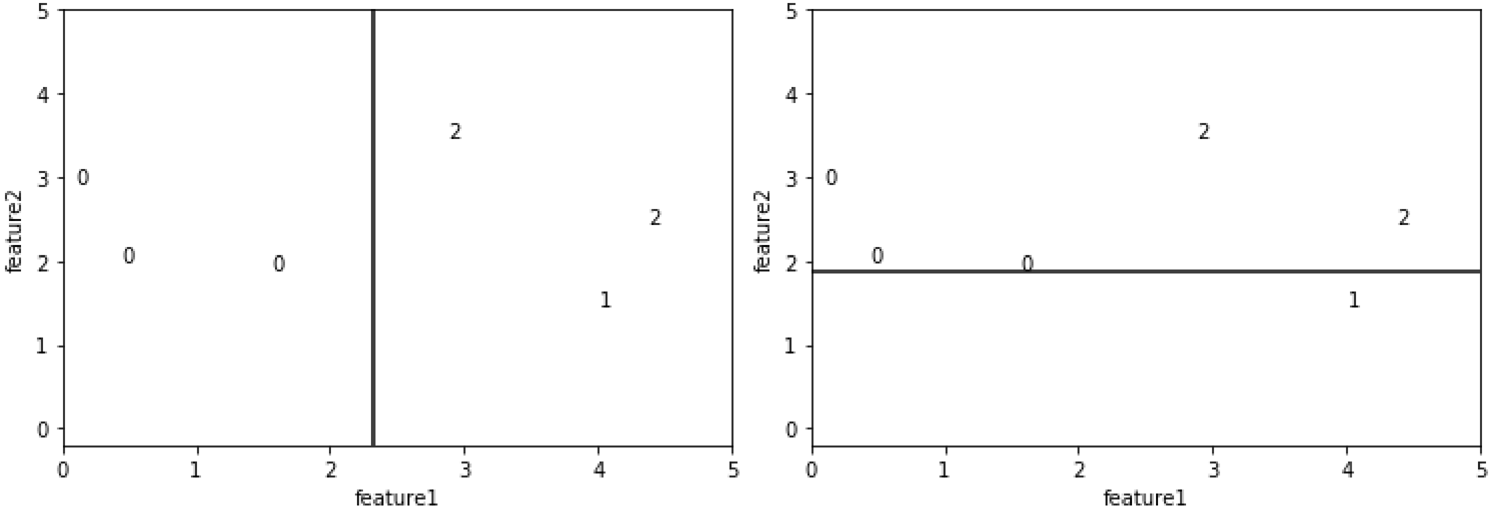
$$= \frac{3}{6} * 0 + \frac{3}{6} * \frac{4}{9} = \frac{4}{18} \approx 0.222$$

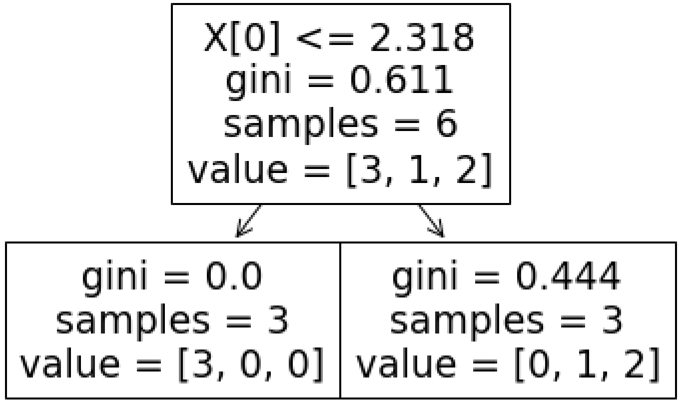


This is the best split if we use feature 1

Best first split

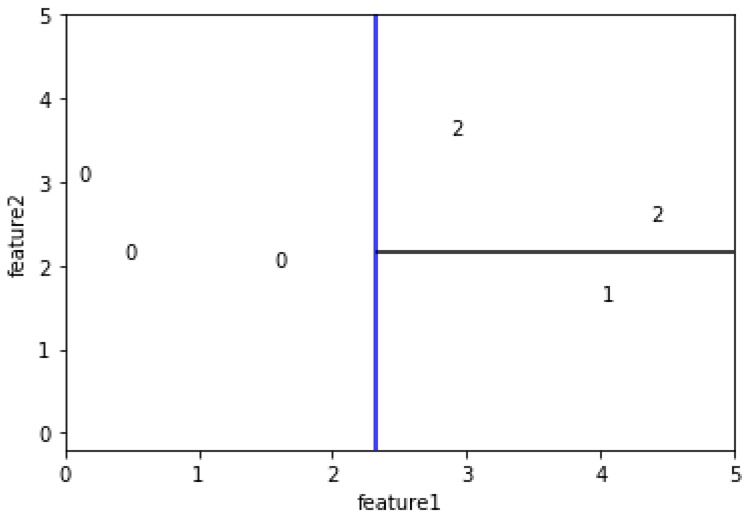
- Best split using feature 1:
  - Left=0
  - Right=0.44
  - Weighted=0.222
- Best split using feature 2:
  - Left=0.375
  - Right=0.5
  - Gini =0.4
- No split:
  - Gini =0.611
- Conclusion:
  - We can split using feature 1 at 2.318
  - This decreases Gini by 0.389,
  - 0.389 is the information gain.

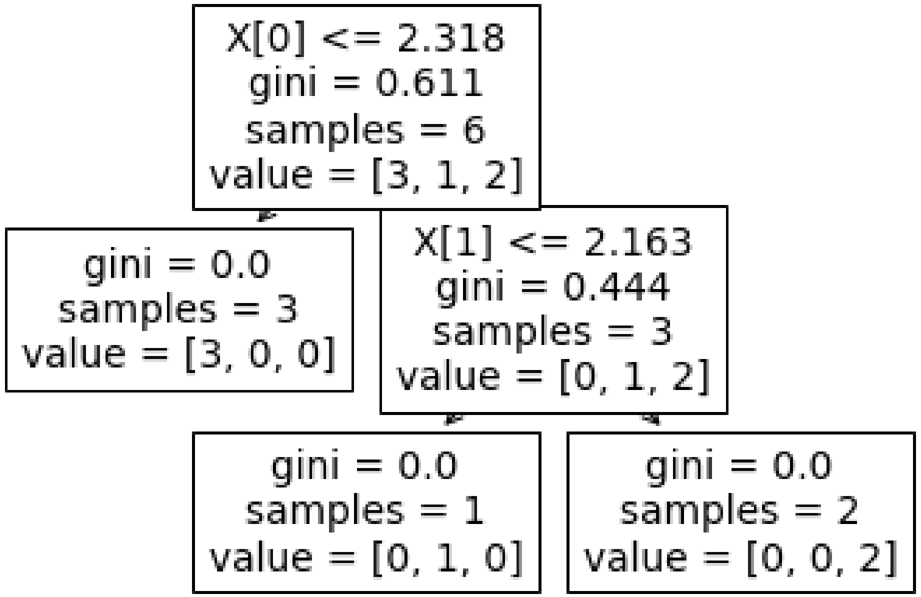




Best second split

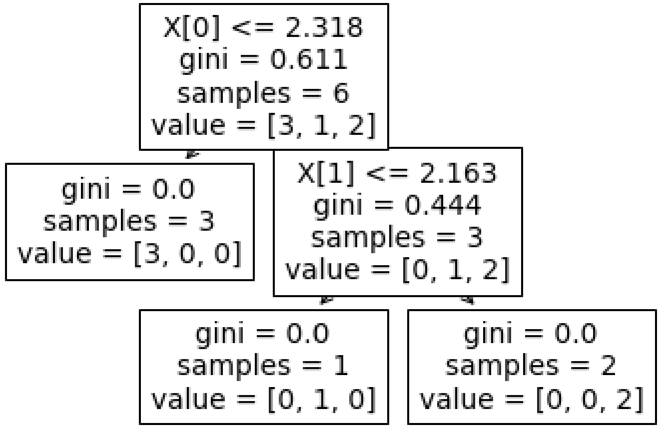
- For the second depth
  - Perform the same computation to decide split
  - Left child of the root
    - Gini is already 0
    - Should not split
  - Right child of the root.
    - Split reduces the Gini index
    - The best split is using feature 2 at x2=2.163

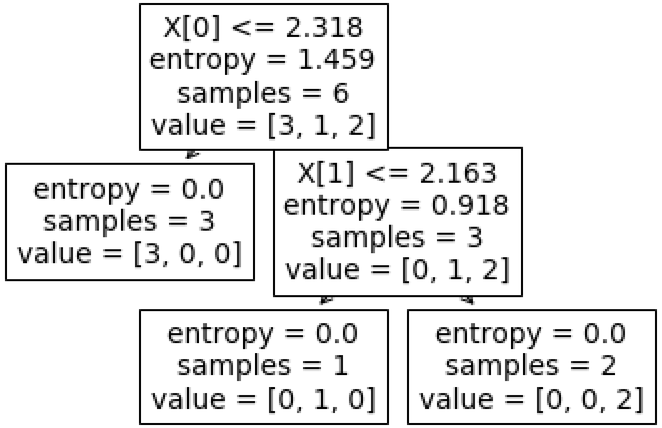




Alternative indicator

- Entropy
  - $-\sum_j p_j \log_2(p_j)$
- Gini-index
  - $1 - \sum_j p_j^2$





Use sklearn to train Decision Tree

- We use `DecisionTreeClassifer` from `sklearn.tree` to train the model
- Three steps
  - Initialize the model
  - Train the model
  - Use model for prediction

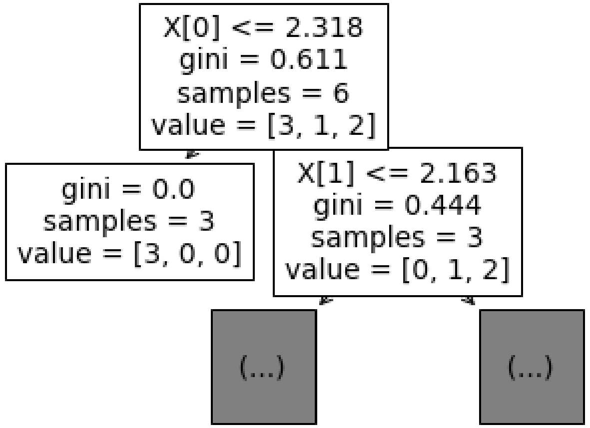
```
from sklearn.tree import DecisionTreeClassifier
## Initialize model
model=DecisionTreeClassifier()
## Train the model
model.fit(X,y)
## Prediction
model.predcit(X)
```

Arguments for `DecisionTreeClassifier`

- DecisionTreeClassifier(*criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1,…*) [source]
  - **criterion*{"gini", "entropy"}***
    - ***default="gini"***
  - **max_depth:**
    - ***int, default=None***
    - If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples
  - **min_samples_split**
    - ***int or float, default=2***
    - The minimum number of samples required to split an internal node
  - **min_samples_leaf**
    - ***int or float, default=1***
    - The minimum number of samples required to be at a leaf node.

Use `plot_tree` to visualize Decision Tree

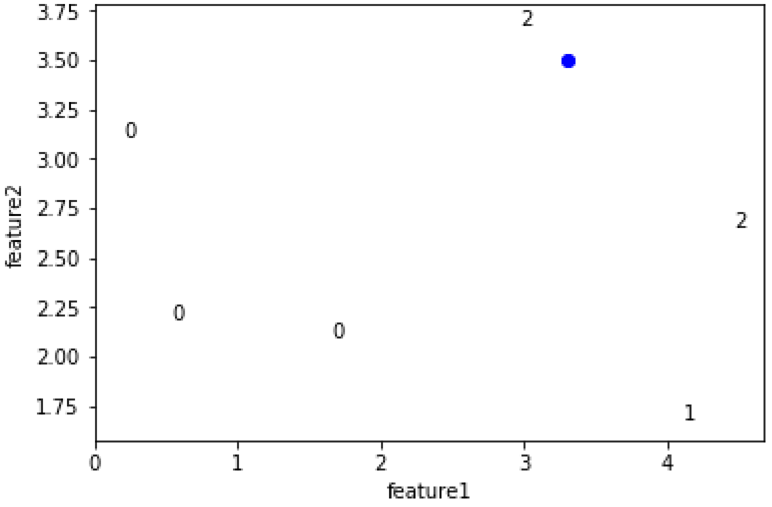
- We use `plot_tree` from `sklearn.tree` to train the model
  - Since a tree can be very large, we might want to only visualize part of the tree. We can use `max_depth` to control how deep we want to visualize the tree
  - In the following demo, even though the tree has depth=2, we visualize only to depth=1




```
from sklearn.tree import plot_tree
## plot tree
plot_tree(Tree,max_depth=1)
plt.show()
```

Model performance

- Accuracy:
  - $Accuracy = \frac{\#correct\ prediction}{\#samples}$
- Error Rate
  - $ErrorRate = 1 - Accuracy$
- Precision: The ratio of how much of the predicted is correct
  - $Precision_j = \frac{\#Predicted\ to\ be\ label\ j\ and\ indeed\ label\ j}{\#Predicted\ to\ be\ label\ j}$
- Recall: The ratio of how many of the actual labels were predicted.
  - $Recall_j = \frac{\#Predicted\ to\ be\ label\ j\ and\ indeed\ label\ j}{\#Total\ number\ of\ label\ j}$
- F1-score:
  - $F1_j = 2\frac{Precision_j \times Recall_j}{Precision_j + Recall_j}$



- Accuracy/Error Rate
  - $Accuracy = \frac{3}{6} = \frac{1}{2}$
  - $ErrorRate = 1 - \frac{1}{2} = \frac{1}{2}$
- Precision/Recall
  - $Precision_0 = \frac{3}{6} = \frac{1}{2}, Recall_0 = \frac{3}{3} = 1$
  - $Precision_1 = \frac{0}{0} = N/A, Recall_1 = \frac{0}{2} = 0$
  - $Precision_2 = \frac{0}{0} = N/A, Recall_2 = \frac{0}{1} = 0$
- F1-score
  - $F1_0 = 2\frac{\frac{1}{2}\times 1}{\frac{1}{2}+1} = \frac{2}{3}, F1_1, F1_2 = N/A$

Check performance of a tree

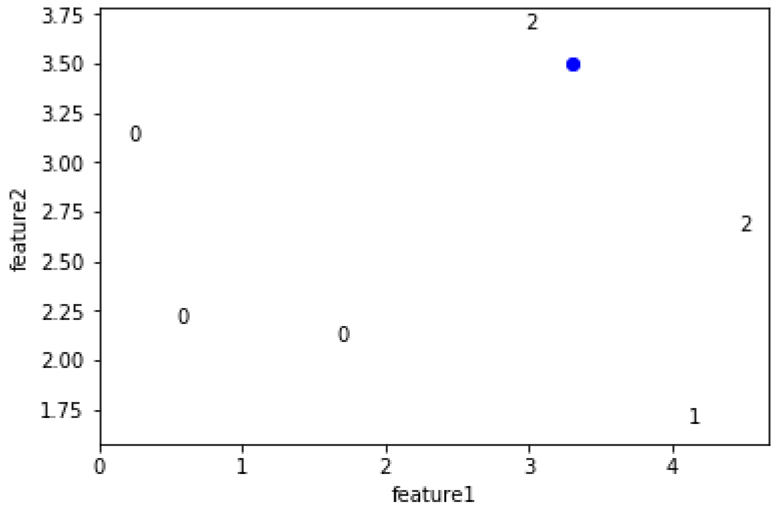
- We can use classification_report from sklearn.metrics to measure the performance of prediction

```
from sklearn.metrics import classification_report
## Initialize model
print(classification_report (y_true, y_pred)
```

```
              precision    recall  f1-score   support

           0       0.50      1.00      0.67         3
           1       0.00      0.00      0.00         1
           2       0.00      0.00      0.00         2

    accuracy                           0.50         6
   macro avg       0.17      0.33      0.22         6
weighted avg       0.25      0.50      0.33         6
```




```
/home/codio/.local/lib/python3.6/site-packages/sklearn/metrics/_classification.py:1
-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/home/codio/.local/lib/python3.6/site-packages/sklearn/metrics/_classification.py:1
-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/home/codio/.local/lib/python3.6/site-packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning: Precision and F
-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Cross-validation

- For Decision Tree classification, if we keep on splitting
  - We can always get 100% accuracy on the dataset if we keep splitting.
  - The model might perform poorly on new data.

- Solution: Split the data into two sets
  - Training (For train the model)
  - Test (For the model performance evaluation)

```
from sklearn.model_selection import train_test_split
# split the features and labels into training (80%) and testing (20%) with a fixed order
X_train, X_val, y_train, y_val = train_test_split(y,X,test_size=0.2, random_state=0)
```

Hyper-parameter tuning

- Cross-validation can be used for hyper-parameter tuning.
- For example, if we want to decide the maximum depth, we can do the following:

```
for MDvalue in maximum_depth:
    Train the model with MDvalue on the training set
    Predict the accuracy on the testing set
Choose best model corresponds to max(MDvalue)
```