Lecture 2: Terminology, Baselines, Decision Trees

UBC 2022 Summer

Instructor: Mehrdad Oveisi

Imports, LOs

Imports

In [1]:
```
import glob
import os
import re
import sys
from collections import Counter, defaultdict

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

sys.path.append("code/.")
import graphviz
import IPython
import mglearn
from IPython.display import HTML, display
from plotting_functions import *
from sklearn.dummy import DummyClassifier
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz
from utils import *

plt.rcParams["font.size"] = 16
pd.set_option("display.max_colwidth", 200)
```

Getting to know you survey

Quick recap: True or False?

- There are different types of machine learning.
- Predicting spam and predicting housing prices are both examples of supervised machine learning.
- For problems such as spelling correction, translation, face recognition, spam identification, if you are a domain expert, it's usually faster and scalable to come up with a robust set of rules manually rather than building a machine learning model.
- Google News is likely be using machine learning to organize news.

Learning outcomes

From this lecture, you will be able to

- identify whether a given problem could be solved using supervised machine learning or not;
- differentiate between supervised and unsupervised machine learning;
- explain machine learning terminology such as features, targets, predictions, training, and error;
- differentiate between classification and regression problems;
- use `DummyClassifier` and `DummyRegressor` as baselines for machine learning problems;
- explain the `fit` and `predict` paradigm and use `score` method of ML models;
- broadly describe how decision tree prediction works;
- use `DecisionTreeClassifier` and `DecisionTreeRegressor` to build decision trees using `scikit-learn`;
- visualize decision trees;
- explain the difference between parameters and hyperparameters;
- explain the concept of decision boundaries;
- explain the relation between model complexity and decision boundaries.

Terminology [video]

You will see a lot of variable terminology in machine learning and statistics. Let's familiarize ourselves with some of the basic terminology used in ML.

Check out the accompanying video on this material.

Big picture and datasets

In this lecture, we'll talk about our first machine learning model: Decision trees. We will also familiarize ourselves with some common terminology in supervised machine learning.

Toy datasets

Later in the course we will use larger datasets from Kaggle, for instance. But for our first couple of lectures, we will be working with the following three toy datasets:

- Quiz2 grade prediction classification dataset
- Quiz2 grade prediction regression dataset
- Canada USA cities dataset

If it's not necessary for you to understand the code, I will put it in one of the files under the `code` directory to avoid clutter in this notebook. For example, most of the plotting code is going to be in `code/plotting_functions.py`.

I'll be using the following grade prediction toy dataset to demonstrate the terminology. Imagine that you are taking a course with four homework assignments and two quizzes. You and your friends are quite nervous about your quiz2 grades and you want to know how will you do based on your previous performance and some other attributes. So you decide to collect some data from your friends from last year and train a supervised machine learning model for quiz2 grade prediction.

In [2]:
```
classification_df = pd.read_csv("data/quiz2-grade-toy-classification.csv")
print(classification_df.shape)
classification_df.head()
```

```
(21, 8)
```

Out[2]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1	1	92	93	84	91	92	A+
1	1	0	94	90	80	83	91	not A+
2	0	0	78	85	83	80	80	not A+
3	0	1	91	94	92	91	89	A+
4	0	1	77	83	90	92	85	A+

Recap: Supervised machine learning

Tabular data

In supervised machine learning, the input data is typically organized in a **tabular** format, where rows are **examples** and columns are **features**. One of the columns is typically the **target**.

Features : Features are relevant characteristics of the problem, usually suggested by experts. Features are typically denoted by X and the number of features is usually denoted by d.

Target : Target is the feature we want to predict (typically denoted by y).

Example : A row of feature values. When people refer to an example, it may or may not include the target corresponding to the feature values, depending upon the context. The number of examples is usually denoted by n.

Training : The process of learning the mapping between the features (X) and the target (y).

Example: Tabular data for grade prediction

The tabular data usually contains both: the features (`X`) and the target (`y`).

In [3]:
```
classification_df = pd.read_csv("data/quiz2-grade-toy-classification.csv")
```

```
classification_df.head()
```

Out[3]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1	1	92	93	84	91	92	A+
1	1	0	94	90	80	83	91	not A+
2	0	0	78	85	83	80	80	not A+
3	0	1	91	94	92	91	89	A+
4	0	1	77	83	90	92	85	A+

So the first step in training a supervised machine learning model is separating `X` and `y`.

In [4]:

```
X = classification_df.drop(columns=["quiz2"])
y = classification_df["quiz2"]
X.head()
```

Out[4]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1
0	1	1	92	93	84	91	92
1	1	0	94	90	80	83	91
2	0	0	78	85	83	80	80
3	0	1	91	94	92	91	89
4	0	1	77	83	90	92	85

In [5]:

```
y.head()
```

Out[5]:

```
0        A+
1    not A+
2    not A+
3        A+
4        A+
Name: quiz2, dtype: object
```

Example: Tabular data for the housing price prediction

Here is an example of tabular data for housing price prediction. You can download the data from here.

In [6]:

```
housing_df = pd.read_csv("data/kc_house_data.csv")
housing_df = housing_df.drop(["id", "date"], axis=1)
housing_df.head()
```

Out[6]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above
0	221900.0	3	1.00	1180	5650	1.0	0	0	3	7	1180
1	538000.0	3	2.25	2570	7242	2.0	0	0	3	7	2170
2	180000.0	2	1.00	770	10000	1.0	0	0	3	6	770
3	604000.0	4	3.00	1960	5000	1.0	0	0	5	7	1050
4	510000.0	3	2.00	1680	8080	1.0	0	0	3	8	1680

In [7]:

```
X = housing_df.drop(columns=["price"])
y = housing_df["price"]
```

```python
X.head()
```

Out[7]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_baser
0	3	1.00	1180	5650	1.0	0	0	3	7	1180	
1	3	2.25	2570	7242	2.0	0	0	3	7	2170	
2	2	1.00	770	10000	1.0	0	0	3	6	770	
3	4	3.00	1960	5000	1.0	0	0	5	7	1050	
4	3	2.00	1680	8080	1.0	0	0	3	8	1680	

In [8]:

```python
y.head()
```

Out[8]:

```
0    221900.0
1    538000.0
2    180000.0
3    604000.0
4    510000.0
Name: price, dtype: float64
```

In [9]:

```python
X.shape
```

Out[9]:

```
(21613, 18)
```

Of course, to a machine, column names (features) have no meaning. Only feature values and how they vary across examples mean something.

Alternative terminology for examples, features, targets, and training

- **examples** = rows = samples = records = instances
- **features** = inputs = predictors = explanatory variables = regressors = independent variables = covariates
- **targets** = outputs = outcomes = response variable = dependent variable = labels (if categorical).
- **training** = learning = fitting

Check out the MDS terminology document.

Supervised learning vs. Unsupervised learning

In **supervised learning**, training data comprises a set of features (X) and their corresponding targets (y). We wish to find a **model function f** that relates X to y. Then use that model function **to predict the targets** of new examples.

In **unsupervised learning** training data consists of observations (X) **without any corresponding targets**.

Unsupervised learning could be used to **group similar things together** in X or to provide **concise summary** of the data. We'll learn more about this topic in later videos.

- Supervised machine learning is about function approximation, i.e., finding the mapping function from `X` to `y`
- Unsupervised machine learning is about concisely describing the data.

Classification vs. Regression

In supervised machine learning, there are two main kinds of learning problems based on what they are trying to predict.

- **Classification problem**: predicting among two or more discrete classes
  - Example1: Predict whether a patient has a liver disease or not
  - Example2: Predict whether a student would get an A+ or not in quiz2.
- **Regression problem**: predicting a continuous value
  - Example1: Predict housing prices
  - Example2: Predict a student's score in quiz2.

In [10]:
```
# quiz2 classification toy data
classification_df = pd.read_csv("data/quiz2-grade-toy-classification.csv")
classification_df.head(4)
```

Out[10]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1	1	92	93	84	91	92	A+
1	1	0	94	90	80	83	91	not A+
2	0	0	78	85	83	80	80	not A+
3	0	1	91	94	92	91	89	A+

In [11]:
```
# quiz2 regression toy data
regression_df = pd.read_csv("data/quiz2-grade-toy-regression.csv")
regression_df.head(4)
```

Out[11]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1	1	92	93	84	91	92	90
1	1	0	94	90	80	83	91	84
2	0	0	78	85	83	80	80	82
3	0	1	91	94	92	91	89	92

❓❓ Questions for you

Exercise 2.1: `X` and `y`

1. How many examples and features are there in the housing price data above? You can use `df.shape` to get number of rows and columns in a dataframe.
2. For each of the following examples what would be the relevant features and what would be the target?
   A. Sentiment analysis
   B. Fraud detection
   C. Face recognition

Exercise 2.1: One solution!

1. Number of examples: 21613, number of features: 18
2. Open-ended

In [12]:
```
housing_df.shape
```

Out[12]:
```
(21613, 19)
```

Exercise 2.2: Supervised vs. unsupervised

Which of these are examples of supervised learning?

1. Finding groups of similar properties in a real estate data set.
2. Predicting real estate prices based on house features like number of rooms, learning from past sales as examples.
3. Grouping articles on different topics from different news sources (something like the Google News app).
4. Detecting credit card fraud based on examples of fraudulent and non-fraudulent transactions.

Exercise 2.2: Solution

- 2 and 4 are examples of supervised machine learning

Exercise 2.3: Classification vs. Regression

Which of these are examples of classification and which ones are of regression?

1. Predicting the price of a house based on features such as number of bedrooms and the year built.
2. Predicting if a house will sell or not based on features like the price of the house, number of rooms, etc.
3. Predicting percentage grade in CPSC 330 based on past grades.
4. Predicting whether you should bicycle tomorrow or not based on the weather forecast.

Exercise 2.3: Solution

- classification: 2, 4
- regression: 1, 3

Baselines [video]

Check out the accompanying video on this material.

Supervised learning (Reminder)

- Training data $\rightarrow$ Machine learning algorithm $\rightarrow$ ML model
- Unseen test data + ML model $\rightarrow$ predictions

Let's build a very simple supervised machine learning model for quiz2 grade prediction problem.

In [13]:
```
classification_df = pd.read_csv("data/quiz2-grade-toy-classification.csv")
classification_df.head()
```

Out[13]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1	1	92	93	84	91	92	A+
1	1	0	94	90	80	83	91	not A+
2	0	0	78	85	83	80	80	not A+
3	0	1	91	94	92	91	89	A+
4	0	1	77	83	90	92	85	A+

In [14]:
```
classification_df['quiz2'].value_counts()
```

Out[14]:
```
not A+    11
A+        10
Name: quiz2, dtype: int64
```

Seems like "not A+" occurs more frequently than "A+". What if we predict "not A+" all the time?

Baselines

Baseline : A simple machine learning algorithm based on simple rules of thumb.

- For example, most frequent baseline always predicts the most frequent label in the training set.
- Baselines provide a way to sanity check your machine learning model.

`DummyClassifier`

- `sklearn`'s baseline model for classification
- Let's train `DummyClassifier` on the grade prediction dataset.

Steps to train a classifier using `sklearn`

1. Read the data
2. Create X and y
3. Create a classifier object
4. `fit` the classifier
5. `predict` on new examples
6. `score` the model

Reading the data

In [15]:
```
classification_df.head()
```

Out[15]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1	1	92	93	84	91	92	A+
1	1	0	94	90	80	83	91	not A+
2	0	0	78	85	83	80	80	not A+
3	0	1	91	94	92	91	89	A+
4	0	1	77	83	90	92	85	A+

Create X and y

- X → Feature vectors
- y → Target

In [16]:
```
X = classification_df.drop(columns=["quiz2"])
y = classification_df["quiz2"]
```

Create a classifier object

- `import` the appropriate classifier
- **Create** an object of the classifier

In [17]:
```
from sklearn.dummy import DummyClassifier # import the classifier

dummy_clf = DummyClassifier(strategy="most_frequent") # Create a classifier object
```

`fit` the classifier

- The "**learning**" is carried out when we call `fit` on the classifier object.

In [18]:
```
dummy_clf.fit(X, y); # fit the classifier
```

`predict` the target of given examples

- We can predict the target of examples by calling `predict` on the classifier object.

In [19]:
```
dummy_clf.predict(X) # predict using the trained classifier
```

Out[19]:
```
array(['not A+', 'not A+', 'not A+', 'not A+', 'not A+', 'not A+',
       'not A+', 'not A+', 'not A+', 'not A+', 'not A+', 'not A+',
       'not A+', 'not A+', 'not A+', 'not A+', 'not A+', 'not A+',
       'not A+', 'not A+', 'not A+'], dtype='<U6')
```

`score` your model

- How do you know how well your model is doing?
- For classification problems, by default, `score` gives the **accuracy** of the model, i.e., proportion of correctly predicted targets.

  $accuracy = \frac{\text{correct predictions}}{\text{total examples}}$

In [20]:
```
print("The accuracy of the model on the training data: %0.3f" % (dummy_clf.score(X, y)))
```

```
The accuracy of the model on the training data: 0.524
```

- Sometimes you will also see people reporting **error**, which is usually $1 - accuracy$
- `score`
  - calls `predict` on `X`
  - compares predictions with `y` (true targets)
  - returns the accuracy in case of classification.

In [21]:
```
print(
    "The error of the model on the training data: %0.3f" %
    (1 - dummy_clf.score(X, y))
)
```

```
The error of the model on the training data: 0.476
```

`fit`, `predict`, and `score` summary

Here is the general pattern when we build ML models using `sklearn`.

In [22]:
```
# Create `X` and `y` from the given data
X = classification_df.drop(columns=["quiz2"])
y = classification_df["quiz2"]

clf = DummyClassifier(strategy="most_frequent") # Create a class object
clf.fit(X, y) # Train/fit the model
print(clf.score(X, y)) # Assess the model

new_examples = [[0, 1, 92, 90, 95, 93, 92], [1, 1, 92, 93, 94, 92]] # two new examples
clf.predict(new_examples) # Predict on some new data using the trained model
```

```
0.5238095238095238
```

Out[22]:
```
array(['not A+', 'not A+'], dtype='<U6')
```

You'll be exploring dummy classifier in your lab!

`DummyRegressor`

You can also do the same thing for regression problems using `DummyRegressor`, which predicts mean, median, or constant value of the training set for all examples.

- Let's build a regression baseline model using `sklearn`.

In [23]:
```python
from sklearn.dummy import DummyRegressor

regression_df = pd.read_csv("data/quiz2-grade-toy-regression.csv") # Read data
X = regression_df.drop(columns=["quiz2"]) # Create `X` and `y` from the given data
y = regression_df["quiz2"]
reg = DummyRegressor() # Create a class object
reg.fit(X, y) # Train/fit the model
reg.score(X, y) # Assess the model
new_examples = [[0, 1, 92, 90, 95, 93, 92], [1, 1, 92, 93, 94, 92]]
reg.predict(new_examples) # Predict on some new data using the trained model
```

Out[23]:
```
array([86.28571429, 86.28571429])
```

- The `fit` and `predict` paradigms similar to classification. The `score` method in the context of regression returns somethings called R^2 score. (More on this in later videos.)
  - The maximum R^2 is 1 for perfect predictions.
  - For `DummyRegressor` it returns the mean of the `y` values.

In [24]:
```python
reg.score(X, y)
```

Out[24]:
```
0.0
```

In [25]:
```python
# DummyRegressor returns the mean of the y values:
[y.mean(), reg.predict(new_examples)]
```

Out[25]:
```
[86.28571428571429, array([86.28571429, 86.28571429])]
```

❓❓ Questions for you

Exercise 2.4

1. Order the steps below to build ML models using `sklearn`.
   - `score` to evaluate the performance of a given model
   - `predict` on new examples
   - Creating a model instance
   - Creating `X` and `y`
   - `fit`
2. `predict` takes only `X` as argument whereas `fit` and `score` take both `X` and `y` as arguments. True or False.
3. Have you ever played 20-questions game? If yes, think about how do you decide what question to ask next?

Exercise 2.4: Solution

1. Ordered steps
   - Creating `X` and `y`
   - Creating a model instance
   - `fit`
   - `predict` on new examples
   - `score` to evaluate the performance of a given model
2. True
3. Open-ended.

Decision trees [video]

Check out the accompanying video on this material.

Writing a traditional program to predict quiz2 grade

- Can we do better than the baseline?
- Forget about ML for a second. If you are asked to write a program to predict whether a student gets an A+ or not in quiz2, how would you go for it?
- For simplicity, let's binarize the feature values.

- Is there a pattern that distinguishes yes's from no's and what does the pattern say about today?
- How about a rule-based algorithm with a number of *if else* statements?

  if class_attendance == 1 and quiz1 == 1:

  ```
  quiz2 == "A+"
  ```
  elif class_attendance == 1 and lab3 == 1 and lab4 == 1:

  ```
  quiz2 == "A+"
  ```
  ...

- How many possible rule combinations there could be with the given 7 binary features?
  - Gets unwieldy pretty quickly

Decision tree algorithm

- A machine learning algorithm to derive such rules from data in a principled way.
- Have you ever played 20-questions game? Decision trees are based on the same idea!
- Let's `fit` a decision tree using `scikit-learn` and `predict` with it.

- Recall that `scikit-learn` uses the term `fit` for training or learning and uses `predict` for prediction.

Building decision trees with `sklearn`

Let's **binarize** our toy dataset for simplicity.

In [26]:
```
classification_df = pd.read_csv("data/quiz2-grade-toy-classification.csv")
X = classification_df.drop(columns=["quiz2"])
y = classification_df["quiz2"]

X_binary = X.copy()
columns = ["lab1", "lab2", "lab3", "lab4", "quiz1"]
for col in columns:
    X_binary[col] = X_binary[col].apply(lambda x: 1 if x >= 90 else 0)

X_binary.head()
```

Out[26]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1
0	1	1	1	1	0	1	1
1	1	0	1	1	0	0	1
2	0	0	0	0	0	0	0
3	0	1	1	1	1	1	0
4	0	1	0	0	1	1	0

In [27]:
```
y.head()
```

Out[27]:
```
0        A+
1    not A+
2    not A+
3        A+
4        A+
Name: quiz2, dtype: object
```

`DummyClassifier` on quiz2 grade prediction toy dataset

In [28]:
```
dummy_clf = DummyClassifier(strategy="most_frequent")
dummy_clf.fit(X_binary, y)
dummy_clf.score(X_binary, y)
```

Out[28]:
```
0.5238095238095238
```

`DecisionTreeClassifier` on quiz2 grade prediction toy dataset

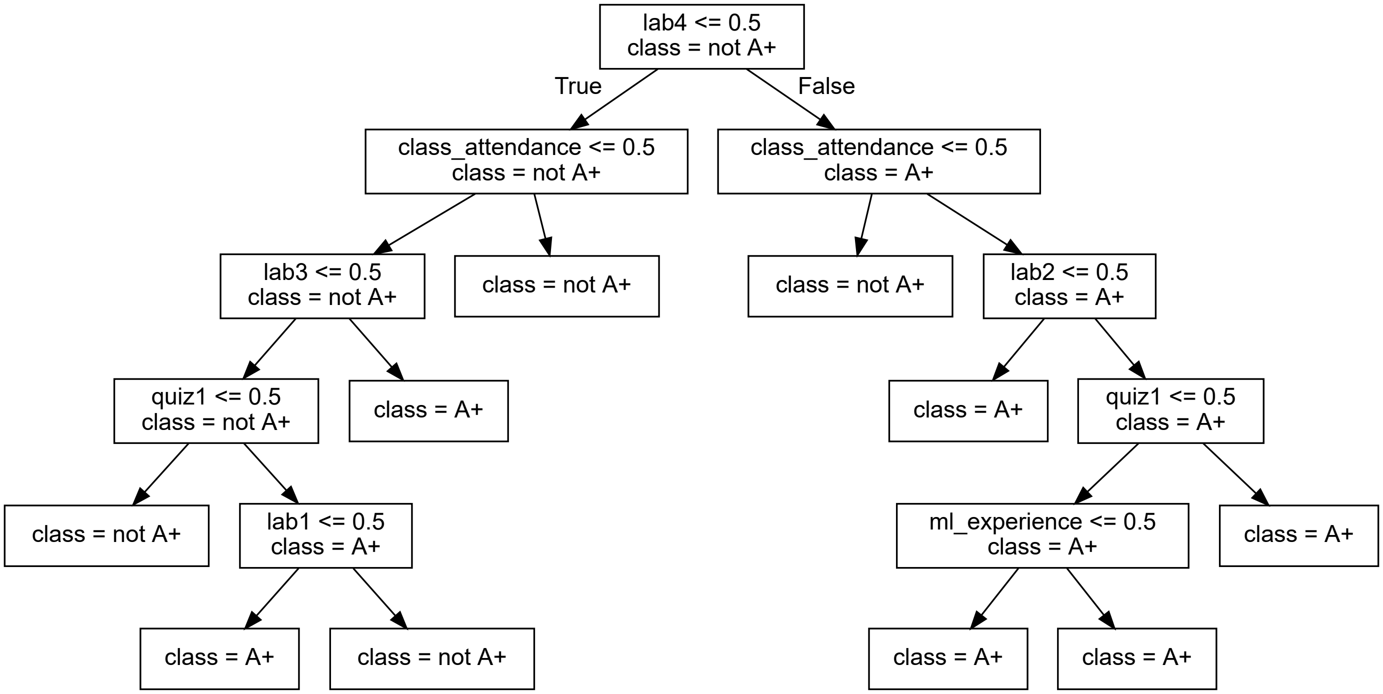
In [29]:
```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier() # Create a decision tree
model.fit(X_binary, y) # Fit a decision tree
model.score(X_binary, y) # Assess the model
```

Out[29]:
```
0.9047619047619048
```

The decision tree classifier is giving much higher accuracy than the dummy classifier. That's good news!

In [30]:
```
display_tree(X_binary.columns, model) # model visualization
```

Out[30]:



Some terminology related to trees

Here is a commonly used terminology in a typical representation of decision trees.

A root node : represents the *first condition* to check or question to ask

A branch : *connects* a node (condition) to the next node (condition) in the tree. Each branch typically represents either true or false.

An internal node : represents conditions *within* the tree

A leaf node : represents the *predicted class/value* when the path from root to the leaf node is followed.

Tree depth : The *number of edges* on the path from the root node to the farthest away leaf node.

How does `predict` work?

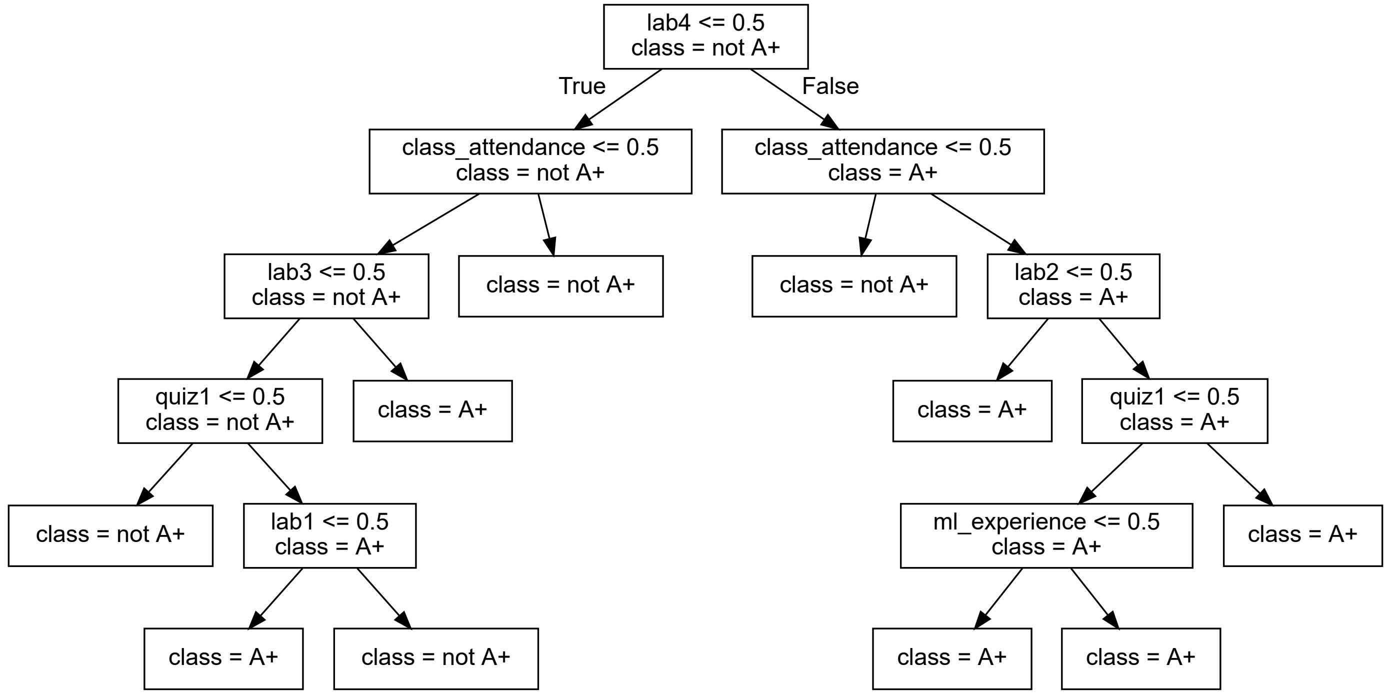
In [31]:
```
new_example = np.array([[0, 1, 0, 0, 1, 1, 1]])
pd.DataFrame(data=new_example, columns=X.columns)
```

Out[31]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1
0	0	1	0	0	1	1	1

In [32]:
```
display_tree(X_binary.columns, model)
```

Out[32]:



What's the prediction for the new example?

In [33]:
```
model.predict(new_example)
```

```
/home/moveisi/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```

Out[33]:
```
array(['A+'], dtype=object)
```

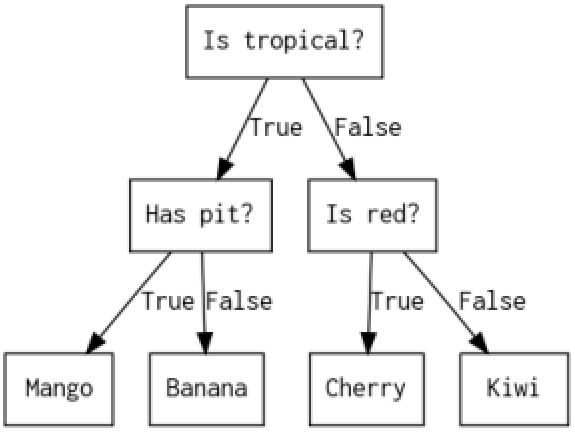
In summary, given a learned tree and a test example, during prediction time,

- Start at the top of the tree. Ask binary questions at each node and follow the appropriate path in the tree. Once you are at a leaf node, you have the prediction.
- Note that the model only considers the features which are in the learned tree and ignores all other features.

How does `fit` work?

- Decision tree is inspired by [20-questions game](#).
- Each node either represents a question or an answer. The terminal nodes (called leaf nodes) represent answers.

In [34]:
```
plot_fruit_tree()
```

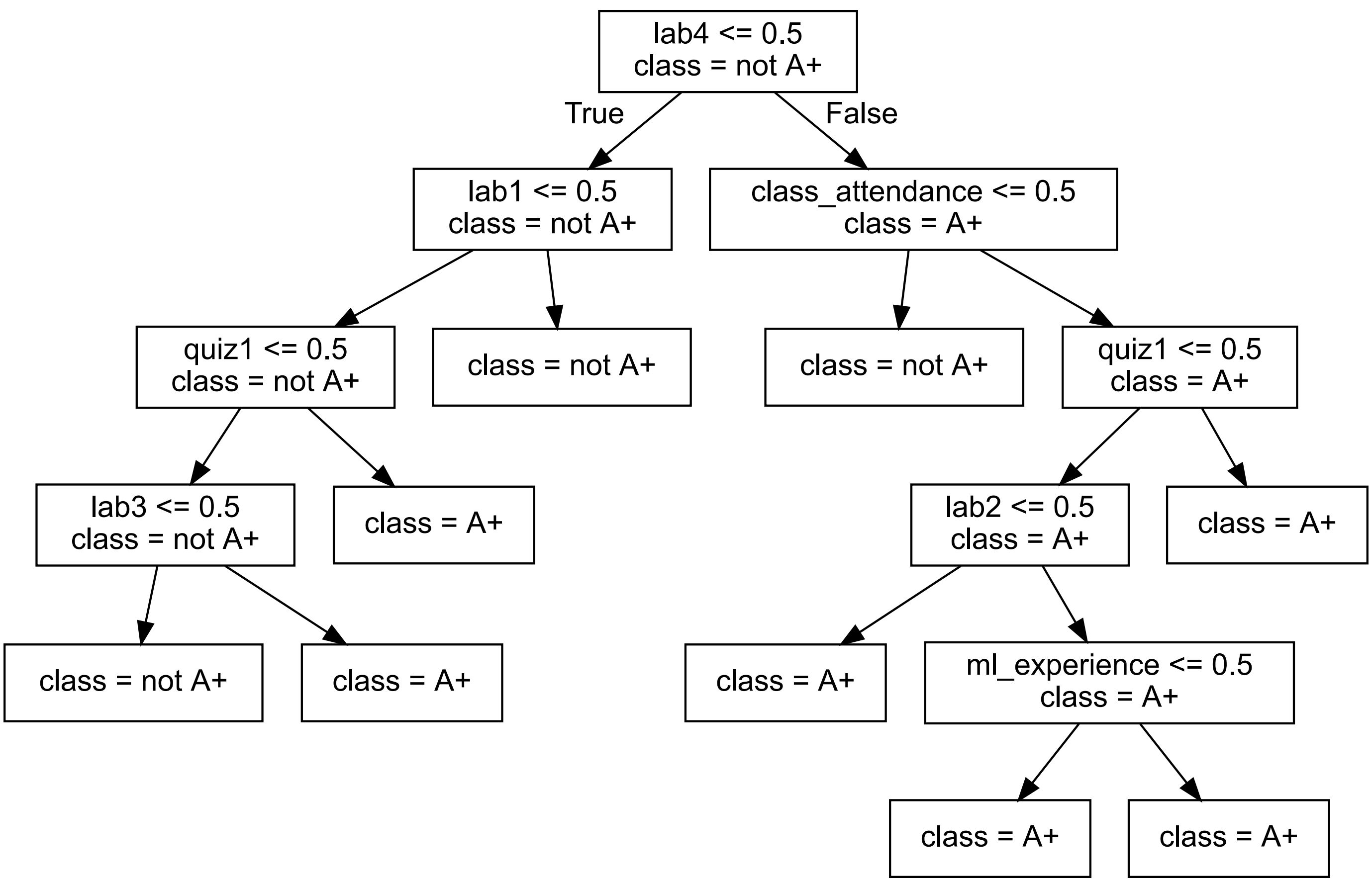


How does `fit` work?

- Which features are most useful for classification?
- Minimize **impurity** at each question
- Common criteria to minimize impurity: gini index, information gain, cross entropy

In [35]:
```python
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier() # Create a decision tree
model.fit(X_binary, y) # Fit a decision tree
display_tree(X_binary.columns, model)
```

Out[35]:



Note

We won't go through **how** it does this - that's CPSC 340.

But it's worth noting that it support two types of inputs:

```
1. Categorical (e.g., Yes/No or more options, as shown in the tree above)
2. Numeric (a number), the decision tree algorithm also picks the *threshold*
```

Decision trees with continuous features

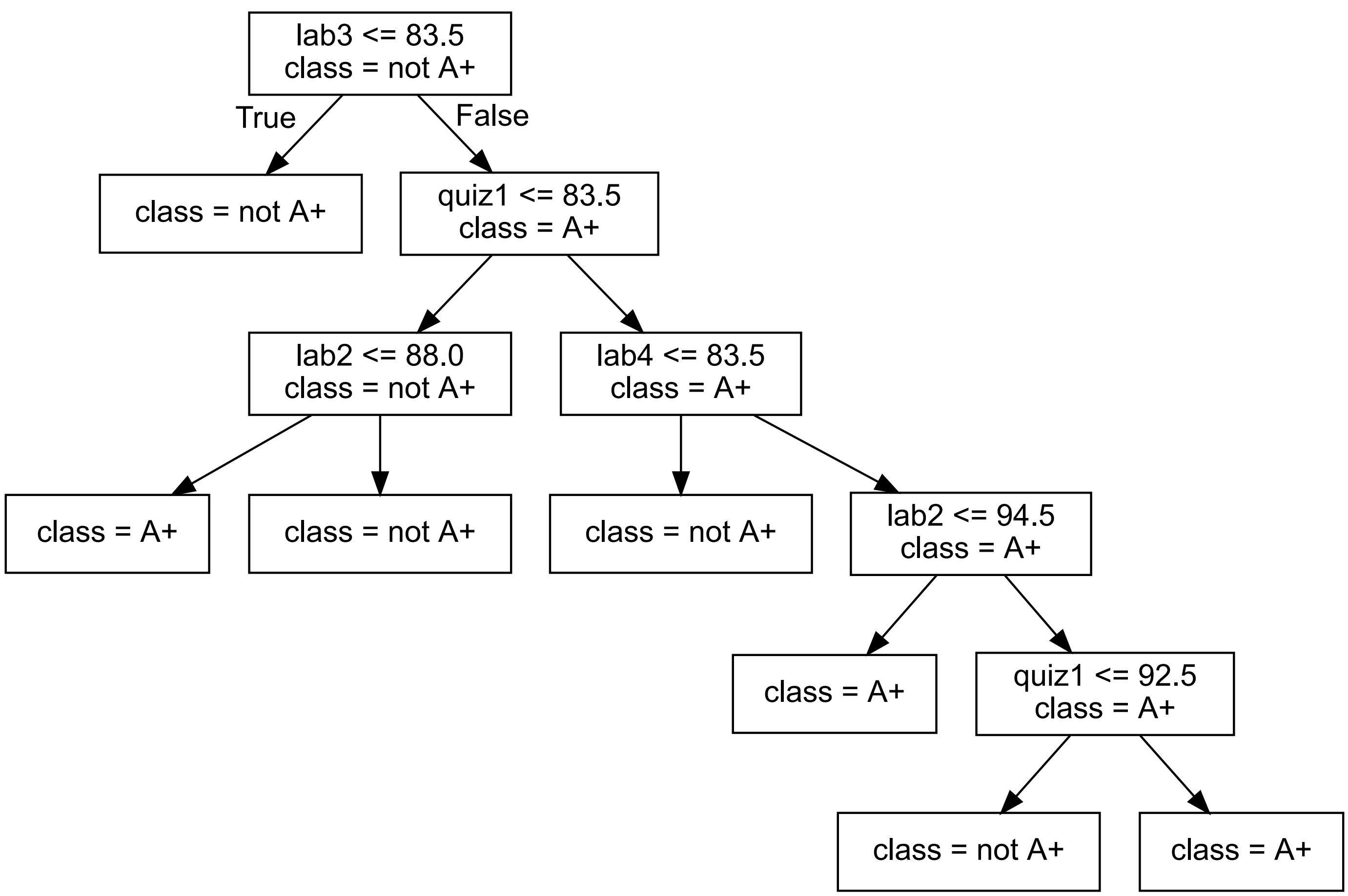
In [36]:
```
X.head()
```

Out[36]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1
0	1	1	92	93	84	91	92
1	1	0	94	90	80	83	91
2	0	0	78	85	83	80	80
3	0	1	91	94	92	91	89
4	0	1	77	83	90	92	85

In [37]:
```
model = DecisionTreeClassifier()
model.fit(X, y)
display_tree(X.columns, model)
```

Out[37]:



Decision tree for regression problems

- We can also use decision tree algorithm for regression.
- Instead of gini, we use some other criteria for splitting. A common one is mean squared error (MSE). (More on this in later videos.)
- `scikit-learn` supports regression using decision trees with `DecisionTreeRegressor`
  - `fit` and `predict` paradigms similar to classification
  - `score` returns somethings called R^2 score.

- The maximum R^2 is 1 for perfect predictions.
- It can be negative which is very bad (worse than `DummyRegressor`).

In [38]:
```
regression_df = pd.read_csv("data/quiz2-grade-toy-regression.csv")
regression_df.head()
```

Out[38]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1	1	92	93	84	91	92	90
1	1	0	94	90	80	83	91	84
2	0	0	78	85	83	80	80	82
3	0	1	91	94	92	91	89	92
4	0	1	77	83	90	92	85	90

In [39]:
```
X = regression_df.drop(["quiz2"], axis=1)
y = regression_df["quiz2"]

depth = 2
reg_model = DecisionTreeRegressor(max_depth=depth)
reg_model.fit(X, y);
regression_df["predicted_quiz2"] = reg_model.predict(X)
print("R^2 score on the training data: %0.3f\n\n" % (reg_model.score(X, y)))
regression_df.head()
```

```
R^2 score on the training data: 0.989
```

Out[39]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2	predicted_quiz2
0	1	1	92	93	84	91	92	90	90.333333
1	1	0	94	90	80	83	91	84	83.000000
2	0	0	78	85	83	80	80	82	83.000000
3	0	1	91	94	92	91	89	92	92.000000
4	0	1	77	83	90	92	85	90	90.333333

❓❓ Questions for you to ponder on

Exercise 2.5

1. Should change in features (i.e., binarizing features above) change `DummyClassifier` predictions?

Exercise 2.5: Solution

1. No. `DummyClassifier` does not look at the features.

Exercise 2.6 True or False

1. For the decision tree algorithm to work, the feature values must be numeric.
2. For the decision tree algorithm to work, the target values must be numeric.
3. The decision tree algorithm creates balanced decision trees.

Exercise 2.6: Solution

1. False
2. False
3. False

Break (5 min)

More terminology [video]

- Parameters and hyperparameters
- Decision boundary

Check out the accompanying video on this material.

Parameters

- The decision tree algorithm primarily **learns two** things:
  - the **best feature** to split on
  - the **threshold** for the feature to split on at each node
- These are called ***parameters*** of the decision tree model.
- When predicting on new examples, we need parameters of the model.

In [40]:
```
classification_df = pd.read_csv("data/quiz2-grade-toy-classification.csv")
X = classification_df.drop(columns=["quiz2"])
y = classification_df["quiz2"]
model = DecisionTreeClassifier()
model.fit(X, y);
```

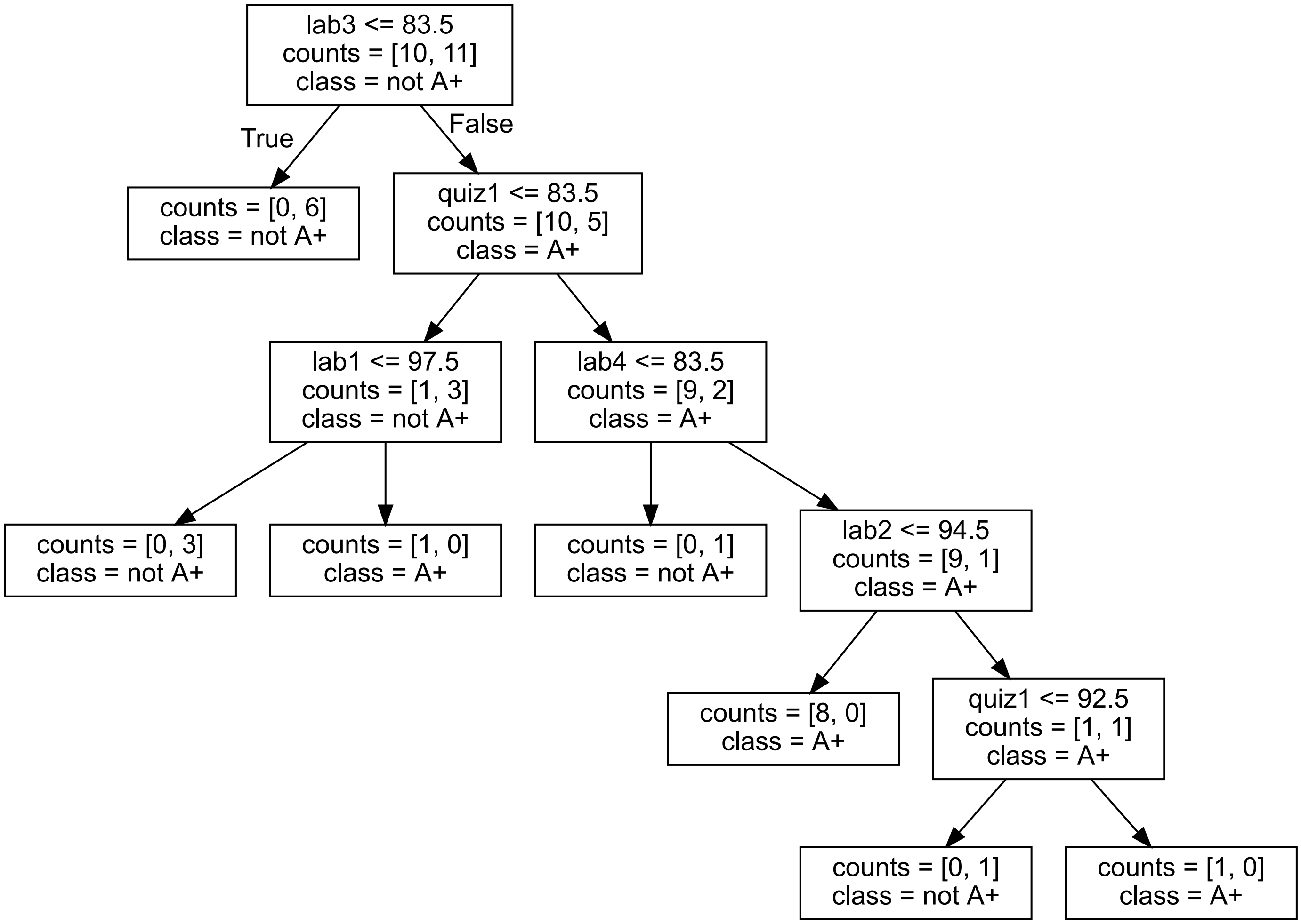
In [41]:
```
X.head()
```

Out[41]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1
0	1	1	92	93	84	91	92
1	1	0	94	90	80	83	91
2	0	0	78	85	83	80	80
3	0	1	91	94	92	91	89
4	0	1	77	83	90	92	85

In [42]:
```
display_tree(X.columns, model, counts=True)
```

Out[42]:



- With the default setting, the nodes are expanded until all leaves are "**pure**".

- The decision tree is creating very specific rules, based on just one example from the data.
- Is it possible to control the learning in any way?
  - Yes! One way to do it is by controlling the **depth** of the tree, which is the length of the longest path from the tree root to a leaf.

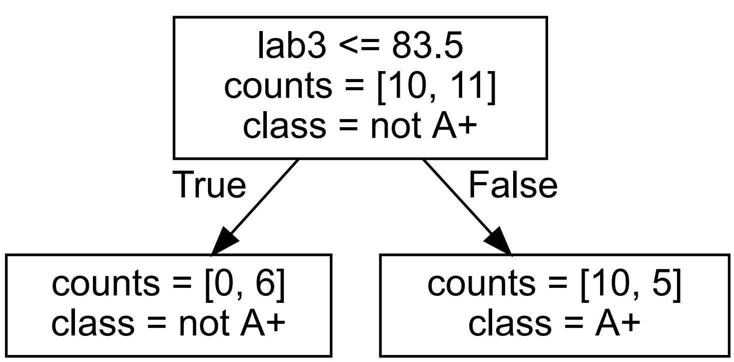
Decision tree with `max_depth=1`

Decision stump : A decision tree with only one split (depth=1) is called a **decision stump**.

In [43]:

```
model = DecisionTreeClassifier(max_depth=1)
model.fit(X, y)
display_tree(X.columns, model, counts=True)
```

Out[43]:



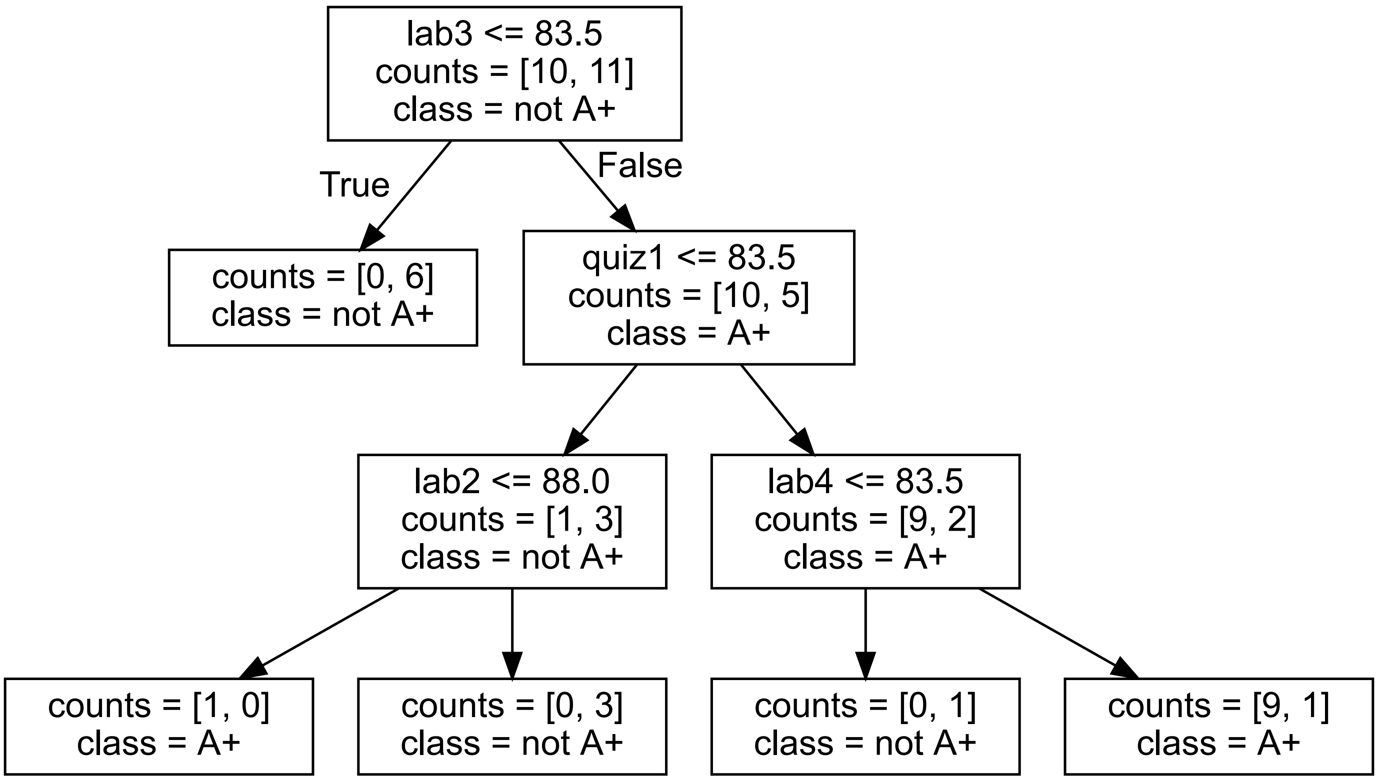
`max_depth` is a **hyperparameter** of `DecisionTreeClassifier`.

Decision tree with `max_depth=3`

In [44]:

```
model = DecisionTreeClassifier(
    max_depth=3
)  # Let's try another value for the hyperparameter
model.fit(X, y)
display_tree(X.columns, model, counts=True)
```

Out[44]:



Parameters and hyperparameters: Summary

Parameters : When you call `fit`, a bunch of values get set, like the features to split on and split thresholds. These are called **parameters**. These are learned by the algorithm from the data during training. We need them during prediction time.

Hyperparameters : Even before calling `fit` on a specific data set, we can set some "knobs" that control the learning. These are called **hyperparameters**. These are specified based on: expert knowledge, heuristics, or systematic/automated optimization (more on this in the coming lectures).

Attention

In `sklearn` hyperparameters are set in the constructor.

Above we looked at the `max_depth` hyperparameter. Some other commonly used hyperparameters of decision tree are:

- `min_samples_split`
- `min_samples_leaf`
- `max_leaf_nodes`

See here for other hyperparameters of a tree.

Decision boundary

What do we do with learned models? So far we have been using them to predict the class of a new instance. Another way to think about them is to ask: what sort of test examples will the model classify as positive, and what sort will it classify as negative?

Example 1: quiz 2 grade prediction

For visualization purposes, let's consider a subset of the data with only two features.

In [45]:
```
X_subset = X[["lab4", "quiz1"]]
X_subset.head()
```

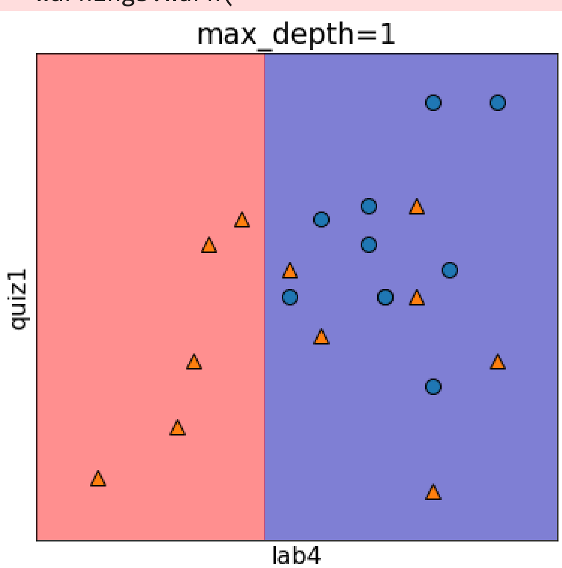
Out[45]:

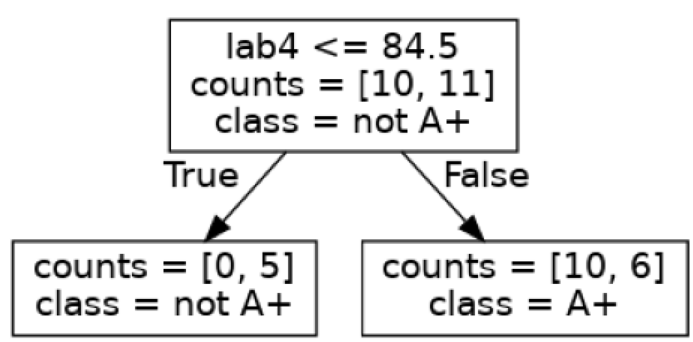
	lab4	quiz1
0	91	92
1	83	91
2	80	80
3	91	89
4	92	85

Decision boundary for `max_depth=1`

In [46]:
```
depth = 1  # decision stump
model = DecisionTreeClassifier(max_depth=depth)
model.fit(X_subset, y)
plot_tree_decision_boundary_and_tree(
    model, X_subset, y, x_label="lab4", y_label="quiz1"
)
```

```
/home/moveisi/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/base.py:450: UserWarn
ing: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature nam
es
  warnings.warn(
```



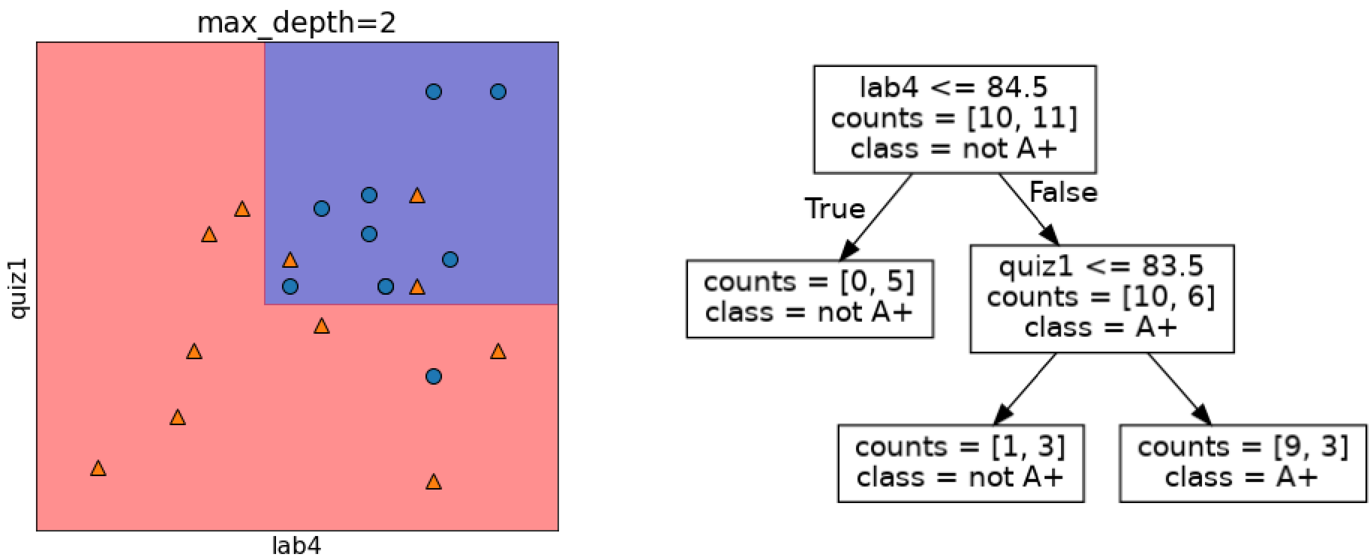


We assume geometric view of the data. Here, the red region corresponds to "not A+" class and blue region corresponds to "A+" class. And there is a line separating the red region and the blue region which is called the **decision boundary** of the model. Different models have different kinds of decision boundaries. In decision tree models, when we are working with only two features, the decision boundary is made up of horizontal and vertical lines. In the example above, the decision boundary is created by asking one question `lab4 <= 84.5`.

Decision boundary for `max_depth=2`

In [47]:
```
model = DecisionTreeClassifier(max_depth=2)
model.fit(X_subset, y)
plot_tree_decision_boundary_and_tree(
    model, X_subset, y, x_label="lab4", y_label="quiz1"
)
```

```
/home/moveisi/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/base.py:450: UserWarn
ing: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature nam
es
  warnings.warn(
```

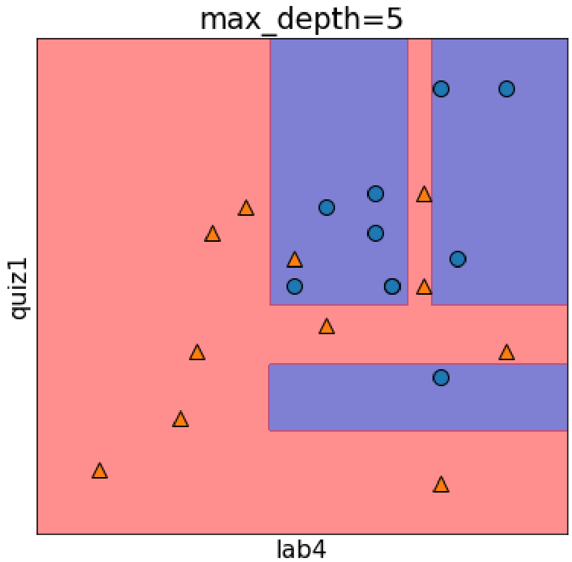


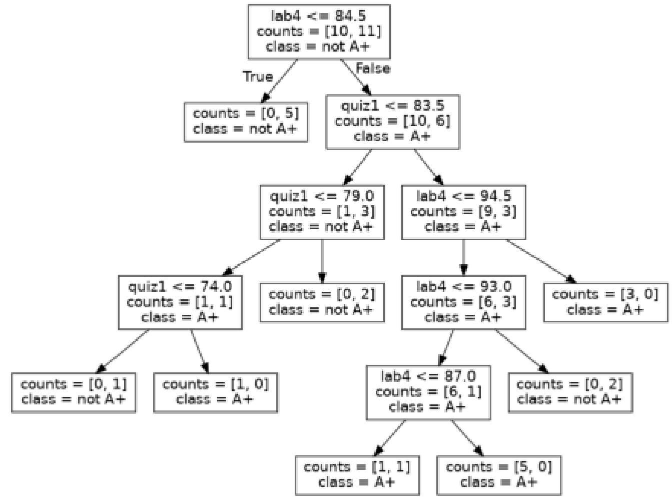
The decision boundary, i.e., the model gets a bit more complicated.

Decision boundary for `max_depth=5`

In [48]:
```
model = DecisionTreeClassifier(max_depth=5)
model.fit(X_subset, y)
plot_tree_decision_boundary_and_tree(
    model, X_subset, y, x_label="lab4", y_label="quiz1"
)
```

```
/home/moveisi/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/base.py:450: UserWarn
ing: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature nam
es
  warnings.warn(
```





The decision boundary, i.e., the model gets even more complicated with `max_depth=5`.

Example 2: Predicting country using the longitude and latitude

Imagine that you are given longitude and latitude of some border cities of USA and Canada along with which country they belong to. Using this training data, you are supposed to come up with a classification model to predict whether a given longitude and latitude combination is in the USA or Canada.

In [49]:
```
### US Canada cities data
df = pd.read_csv("data/canada_usa_cities.csv")
df
```

Out[49]:

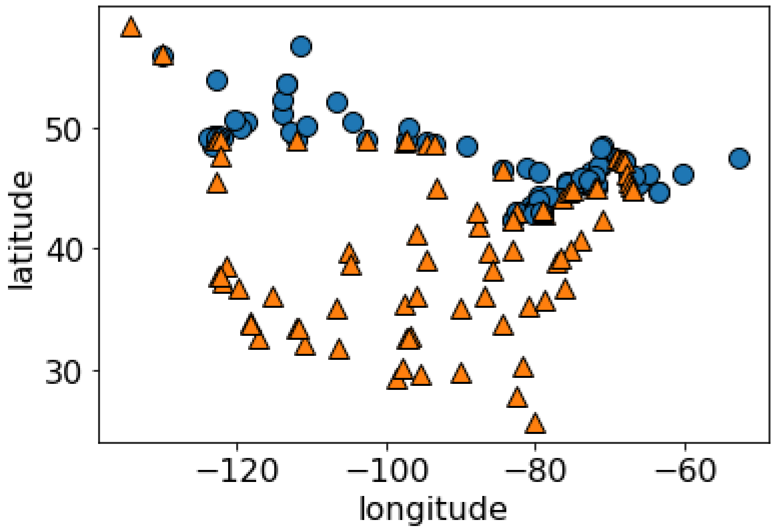
	longitude	latitude	country
0	-130.0437	55.9773	USA
1	-134.4197	58.3019	USA
2	-123.0780	48.9854	USA
3	-122.7436	48.9881	USA
4	-122.2691	48.9951	USA
...	...	...	...
204	-72.7218	45.3990	Canada
205	-66.6458	45.9664	Canada
206	-79.2506	42.9931	Canada
207	-72.9406	45.6275	Canada
208	-79.4608	46.3092	Canada

209 rows × 3 columns

In [50]:
```
X = df[["longitude", "latitude"]]
```

In [51]:
```
y = df["country"]
```

In [52]:
```
mglearn.discrete_scatter(X.iloc[:, 0], X.iloc[:, 1], y)
plt.xlabel("longitude")
plt.ylabel("latitude");
```



Real boundary between Canada and USA

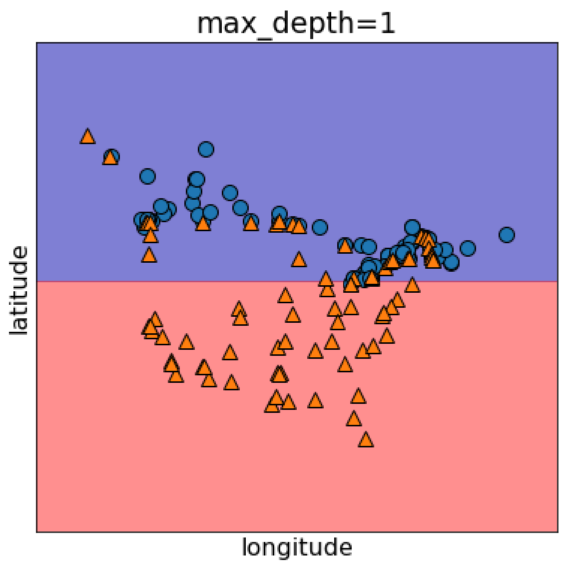
In real life we know what's the boundary between USA and Canada.

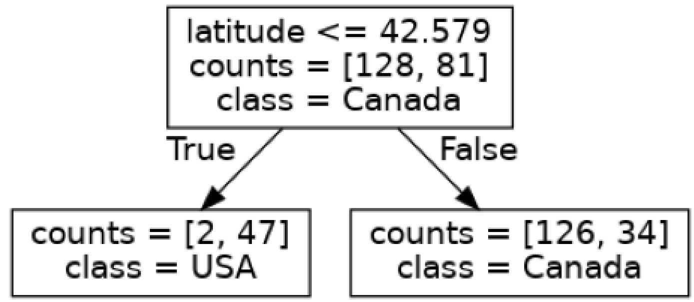
[Source]

Here we want to pretend that we do not know this boundary and we want to infer this boundary based on the limited training examples given to us.

In [53]:
```
model = DecisionTreeClassifier(max_depth=1)
model.fit(X, y)
plot_tree_decision_boundary_and_tree(
    model,
    X,
    y,
    height=6,
    width=16,
    eps=10,
    x_label="longitude",
    y_label="latitude",
)
```

```
/home/moveisi/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/base.py:450: UserWarn
ing: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature nam
es
  warnings.warn(
```

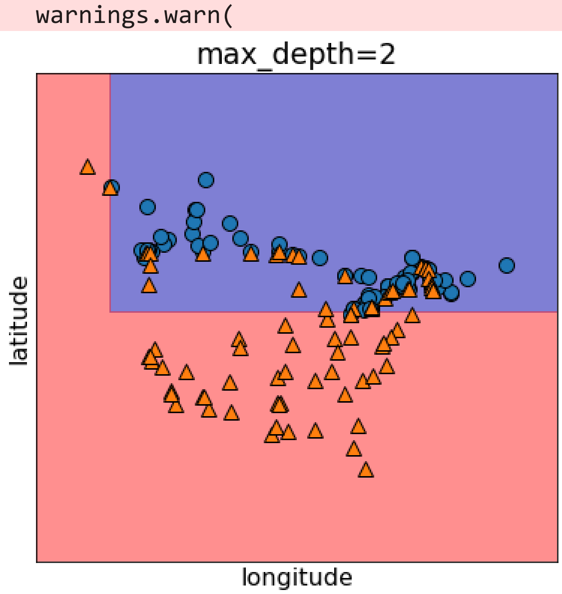


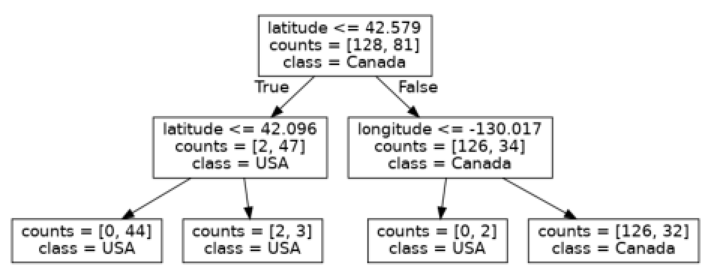


In [54]:

```
model = DecisionTreeClassifier(max_depth=2)
model.fit(X, y)
plot_tree_decision_boundary_and_tree(
    model,
    X,
    y,
    height=6,
    width=16,
    eps=10,
    x_label="longitude",
    y_label="latitude",
)
```

```
/home/moveisi/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/base.py:450: UserWarn
ing: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature nam
es
  warnings.warn(
```





Practice exercises

- If you want more practice, check out module 2 in this online course. All the sections **without** video or notes symbol are exercises.

Attention

If all of you are working on the exercises, especially coding exercises, at the same time, you might have to wait for the real-time feedback for a long time or you might even get an error. There is no solution for this other than waiting for a while and trying it again.

Some background on the online course above: This course is designed by Hayley Boyce, Mike Gelbart, and Varada Kolhatkar. It'll be a great optional resource at the beginning of this class, as it give you a chance to practice what we learn and the framework will provide you real-time feedback.

Final comments, summary, and reflection

What did we learn today?

- There is a lot of terminology and jargon used in ML. Some of the basic terminology includes:
  - Features, target, examples, training
  - Supervised vs. Unsupervised machine learning
  - Classification and regression
  - Accuracy and error
  - Parameters and hyperparameters
  - Decision boundary

- Baselines and steps to train a supervised machine learning model
  - Baselines serve as reference points in ML workflow.

- Decision trees
  - are models that make predictions by sequentially looking at features and checking whether they are above/below a threshold
  - learn a hierarchy of if/else questions, similar to questions you might ask in a 20-questions game.
  - learn axis-aligned decision boundaries (vertical and horizontal lines with 2 features)
  - One way to control the complexity of decision tree models is by using the depth hyperparameter (`max_depth` in `sklearn`).