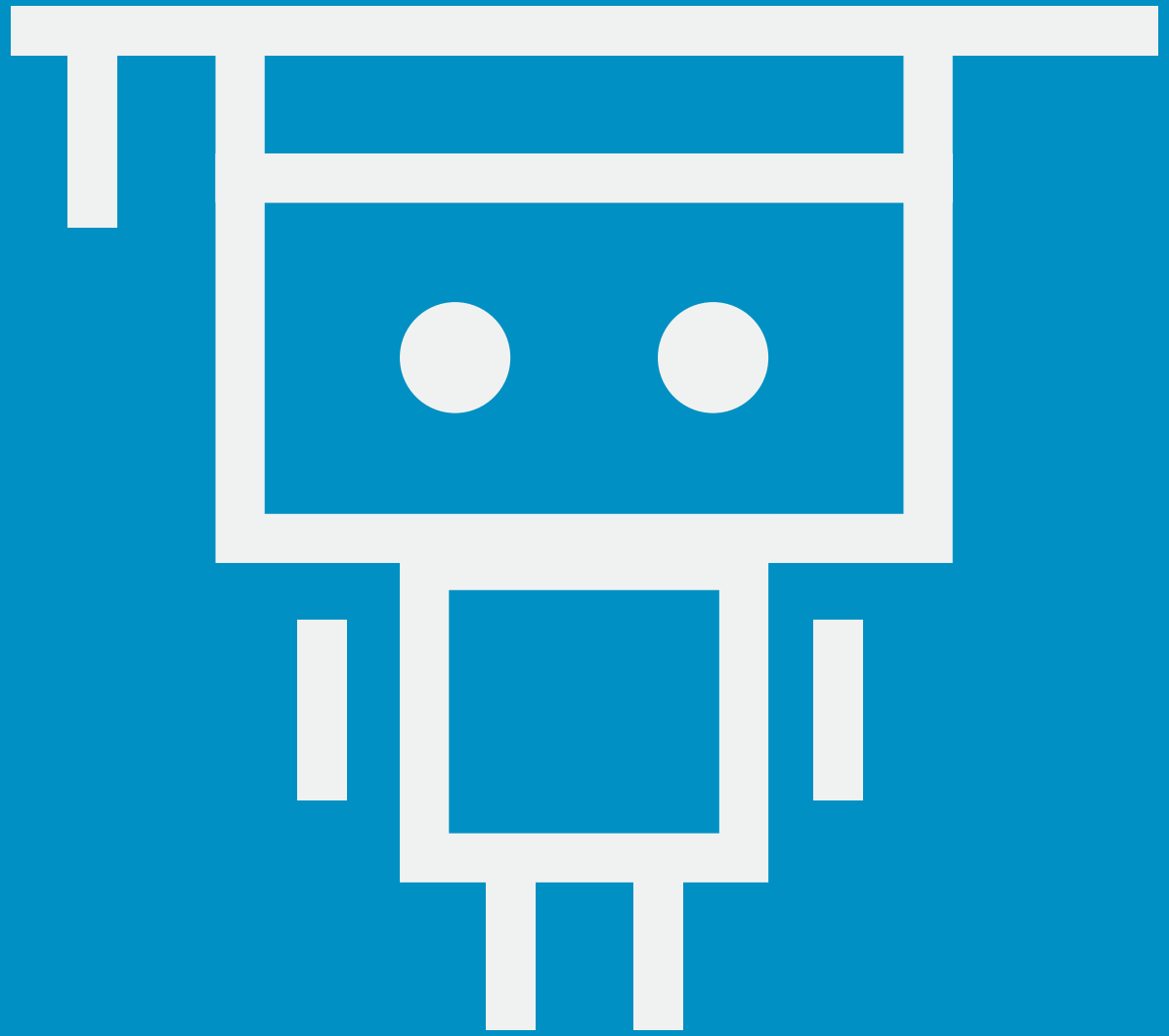


Course Description

- Covers techniques for managing data throughout an end-to-end ML process
- Learn statistical analysis and visualization techniques, including methods for detecting and remedying overfitting
- Gain familiarity with tools in standard ML and data processing libraries

Course Description

- **Lecture 1:** ML Life-cycle, Data Analysis and Basic ML model: K Nearest Neighbors
- **Lecture 2:** Feature Engineering, Decision Trees, Hyper-parameters and AWS Sagemaker
- **Lecture 3:** Optimization, Regression Models, Boosting and Neural Networks

Final Project-Completion

- Apply and experiment with a real-world Amazon dataset.
- Completion of this course is based on your Leaderboard submission.
- Submission page: https://leaderboard.corp.amazon.com/tasks/478
- Submission is open until Saturday 5:00 PM (PST)

Top 3 submissions - this week

Completions

Final Project-Completion

You will get a score after each submission. You can improve your score by making multiple submissions (no upper limit on number of submissions).

- **Requirements:** At least **ONE** submission is **REQUIRED**
- **Non-completion:** Submit 0 models to the Class Leaderboard

After completion, student and manager receive confirmation email.

Topics for today

- Data Imputation and Scaling
- Feature Engineering
- Trees and Ensemble Learning
- Hyperparameter tuning
- Sagemaker
- Final Project

Topics for today

- Data Imputation and Scaling
- Feature Engineering
- Trees and Ensemble Learning
- Hyperparameter tuning
- Sagemaker
- Final Project

Imputing Missing Values

- **Average imputation:** Replaces missing values with the average value in the column. Useful for numeric variables. (We did this yesterday too)

  df['col_name'].fillna((df['col_name'].mean()), inplace=True)

- **Common point imputation:** Use the most common value for that column to replace missing values. Useful for categorical variables.

  df['col_name'].fillna((df['col_name'].mode()), inplace=True)

- **Advanced imputation:** We can learn to predict missing values from complete samples using some machine learning techniques.
  - For example: **AWS Datawig** tool uses neural networks to predict missing values in tabular data. https://github.com/awslabs/datawig

Feature Scaling

- **Motivation:** Many algorithms are sensitive to features being on different scales, e.g., gradient descent and kNN
- **Solution:** Bring features on to the same scale
- **Note:** Some algorithms like decision trees and random forests aren't sensitive to features on different scales
- Common choices (both linear):
  - Mean/variance standardization
  - MinMax scaling

Mean/variance standardization

- Scale values to be centered around mean 0 with standard deviation 1
- Transform: $x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$
- scikit-learn: sklearn.preprocessing.StandardScaler
- Advantages
  - Many algorithms behave better with smaller values
  - Keeps outlier information, but reduces impact

```
from sklearn.preprocessing import StandardScaler
stdsc = StandardScaler()

raw_data = pd.Series([-2.1, 3.4, 0, 6.7, 7.1, 5.4, 0, 0, -5.3, 3.2])
scaled_data = stdsc.fit_transform(raw_data)
print(scaled_data)
```

```
[-1.03503738  0.40981175 -0.48336771  1.27672124  1.38180117  0.93521144
 -0.48336771 -0.48336771 -1.87567688  0.35727179]
```

Use **fit_transform()** only with your training data. Use **transform()** for your test and validation data.

MinMax Scaling

- Scale values so that minimum value is 0 and maximum value is 1
- Transform: $x_{norm}^{(i)} = \frac{x^{(i)} - x_{min}}{x_{max} - x_{min}}$
- scikit-learn: sklearn.preprocessing.MinMaxScaler
- Advantages
  - Robust to small standard deviations

```
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler()

raw_data = pd.Series([-2.1, 3.4, 0, 6.7, 7.1, 5.4, 0, 0, -5.3, 3.2])
scaled_data = mms.fit_transform(raw_data)
print(scaled_data)
```

```
[ 0.25806452  0.7016129   0.42741935  0.96774194  1.          0.86290323
  0.42741935  0.42741935  0.          0.68548387]
```

Use **fit_transform()** only with your training data. Use **transform()** for your test and validation data.

Topics for today

- Data Imputation and Scaling
- Feature Engineering
- Trees and Ensemble Learning
- Hyperparameter tuning
- Sagemaker
- Final Project

Data Cleaning and Tidying

- **Duplicate records:** It might be useful to find and eliminate duplicate records as they are not automatically handled.

```
# Get duplicate rows
dup_records = df[df.duplicated()]
```

```
# Drop duplicate rows
df.drop_duplicates(inplace=True)
```

- **Changing column types:** We can assign different types to data frame columns.

```
df["col1"] = df["col1"].astype('float64')
```

Cleaning Text Data

- Different types of data require special cases of cleaning
- Text is a common data type such as **titles, names, reviews or any freeform input**
- **Motivation:** Messy text harder to find patterns in. Normalize text by removing "noise"
- **Example**: The following two sentences have similar meaning but may seem quite different to a text classifier (i.e. sentiment detector):
  - "The countess (Rebecca) considers\n the boy to be quite naïve."
  - "countess rebecca considers boy naive"

Common Text Preprocessing Steps

- Lowercase
  - text.lower()
- Strip leading/trailing whitespace
  - text.strip()
- Replace punctuation with space
  - re.compile('[%s]' % re.escape(string.punctuation)).sub(' ', text)
- Collapse adjacent spaces and tabs to one space
- Remove or replace special characters (e.g., naïve → naive)
- Remove markup (e.g., HTML)

Lexicon-Based Text Preprocessing

- Stop-word removal: ‘a’, ‘the’, ‘and’, ‘is’, etc.
- Stemming: ‘jumping’, ‘jumped’ → ‘jump’
- Spelling correction: ‘thier’ → ‘their’
- Synonym normalization: ‘awesome’, ‘wonderful’, ‘great’ → ‘great’
- Natural Language Toolkit (NLTK) library
  - Provides support for many of these and more
  - http://www.nltk.org/
- Motivations (justifications):
  - Reducing noise:
    - large quantity without much specific information
  - Boosting signal:
    - Consolidating the strength of the meaning instead of the word choice

Text Features – Bag of Words

- Bag of Words model converts text data into numerical features. We get a number for each word. It can be number of occurrences or binary (present or not)
- Example:
  - sentence_1 = “This is the first document”
  - sentence_2 = “This is the second document”
  - sentence_3 = “and the third one”

vocabulary = {and, document, first, is, one, second, the, third, this}

	and	document	first	is	one	second	the	third	this
sentence_1	0	1	1	1	0	0	1	0	1
sentence_2	0	1	0	1	0	1	1	0	1
sentence_3	1	0	0	0	1	0	1	1	0

Hands-on Text Preprocessing

- We will do the following in this notebook:
  - Text cleaning
  - Preprocessing
  - Getting Bag of Words features

- Open this notebook: https://eider.corp.amazon.com/sazaracs/notebook/NBQM6DD1RI0P

Feature Engineering

- Feature engineering: Using domain and data knowledge to create novel features as inputs for ML models Often more art than science
- Some rules of thumb
  - Use intuition: “What information would *a human* use to predict?”
  - Try generating many features, then apply dimensionality reduction if needed
  - Consider transformations of attributes (e.g., squaring)
  - Consider combinations of attributes (e.g., multiplication)
  - Do not overthink or include too much manual logic
- scikit-learn: sklearn.feature_extraction

Processing Categorical Variables

- Categorical (also called discrete): These variables don't have a natural numerical representation.
  - Example: color ∈ {green, red, blue}, isFraud ∈ {false, true}
- Most Machine Learning models require converting categorical variables to numerical ones.
- Label encoding: Assign a number to each category.
- Things to consider:
  - **Ordinal:** Categories are ordered, e.g., size ∈ {L > M > S}, We can assign L->3, M->2, S->1
  - **Nominal:** Categories are unordered, e.g., color, We can assign the numbers randomly

Encoding Categorical Variables

1.1 Frist let's create a small dataset

```
import pandas as pd
df = pd.DataFrame([
    ['green', 'S', 10.1, 'shirt'],
    ['red', 'M', 13.5, 'pants'],
    ['blue', 'L', 15.3, 'shirt']])
df.columns = ['color', 'size', 'price', 'classlabel']
df
```

	color	size	price	classlabel
0	green	S	10.1	shirt
1	red	M	13.5	pants
2	blue	L	15.3	shirt

Encoding Categorical Variables

1.2 Use sklearn.preprocessing.LabelEncoder for ordinals or lables

It will map categoricals to integer 0 to number of categories - 1

```
from sklearn.preprocessing import LabelEncoder
class_le = LabelEncoder()
y = class_le.fit_transform(df['classlabel'].values)
y
```

```
array([1, 0, 1], dtype=int64)
```

1.3 When LabelEncoder is not what we want to map, we can define our own mapping

```
size_mapping = { 'L': 5, 'M': 2, 'S': 1 }
df['size2'] = df['size'].map(size_mapping)
df
```

	color	size	price	classlabel	size2
0	green	S	10.1	shirt	1
1	red	M	13.5	pants	2
2	blue	L	15.3	shirt	5

Wrong Solution when there is no relationship between categories with more than two categories (binary OK).

Encoding Categorical Variables

- Problem: Encoding nominals with integers is wrong because the ordering and size of the integers is meaningless

- One-hot encoding is better: Explode nominal attribute into many binary attributes, one for each discrete value

- Actually, one for each value except a baseline value.

Encoding Nominals

1.4 Enconding Norminals

Using OneHotEncoder from sklearn

```
from sklearn.preprocessing import OneHotEncoder

X = df[['color', 'size2', 'price']].values
color_le = LabelEncoder()
X[:, 0] = color_le.fit_transform(X[:, 0])
ohe = OneHotEncoder(categorical_features=[0])
X = ohe.fit_transform(X)
X.toarray()
```

```
array([[  0. ,   1. ,   0. ,   1. ,  10.1],
       [  0. ,   0. ,   1. ,   2. ,  13.5],
       [  1. ,   0. ,   0. ,   5. ,  15.3]])
```

Using get_dummies() from pandas

```
pd.get_dummies(df[['color', 'size2', 'price']],
               columns=['color'])
```

	size2	price	color_blue	color_green	color_red
0	1	10.1	0	1	0
1	2	13.5	0	0	1
2	5	15.3	1	0	0

Encoding with many classes

- Define a hierarchy structure:
  - For example: If we a **zip code** column, then we can try to use regions -> states -> city as the hierarchy and you can choose a specific level to encode the zip code column
- We can also try to group the levels into fewer groups by similarity

- **Target Encoding:** Averaging the target value for each category. Then, replace the categorical values with the average target value.

Encoding with many classes

Target Encoding: Averaging the target value for each category. Then, replace the categorical values with the average target value.

Example:

x_1	x_2	y
a	d	1
a	e	1
b	d	0
a	e	0
a	e	0
a	e	1
b	e	0

x_1 -> cat a -> 0.6
x_1 -> cat b -> 0
x_2 -> cat d -> 0.5
x_2 -> cat e -> 0.4

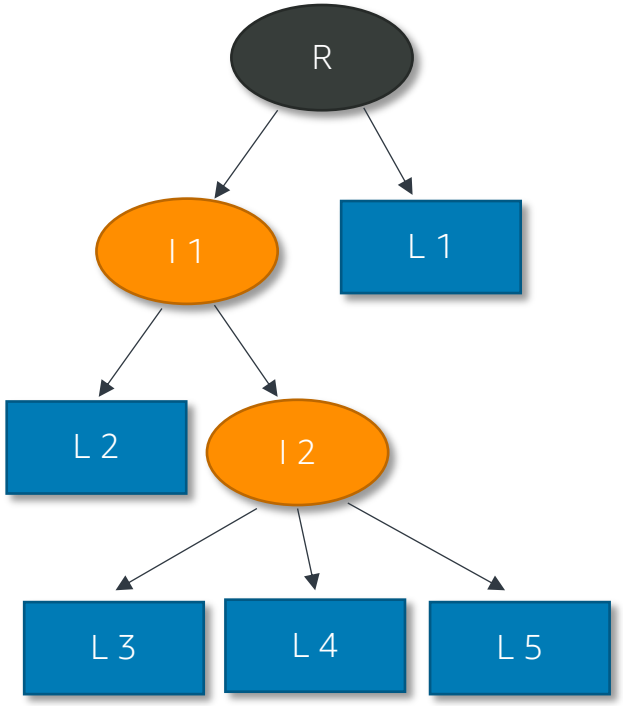
x_1	x_2	y
0.6	0.5	1
0.6	0.4	1
0	0.5	0
0.6	0.4	0
0.6	0.4	0
0.6	0.4	1
0	0.4	0

Topics for today

- Data Imputation and Scaling
- Data Labeling
- Feature Engineering
- Trees and Ensemble Learning
- Hyperparameter tuning
- Sagemaker
- Final Project

Decision Trees

- Decision Trees are flowchart-like structures that can be used for classification or regression tasks.



Root Node:

- It is the start node

Internal Node:

- It has exactly one incoming node and two or more outgoing edges.
- It has attribute conditions to separate records.

Leaf or Terminal Node:

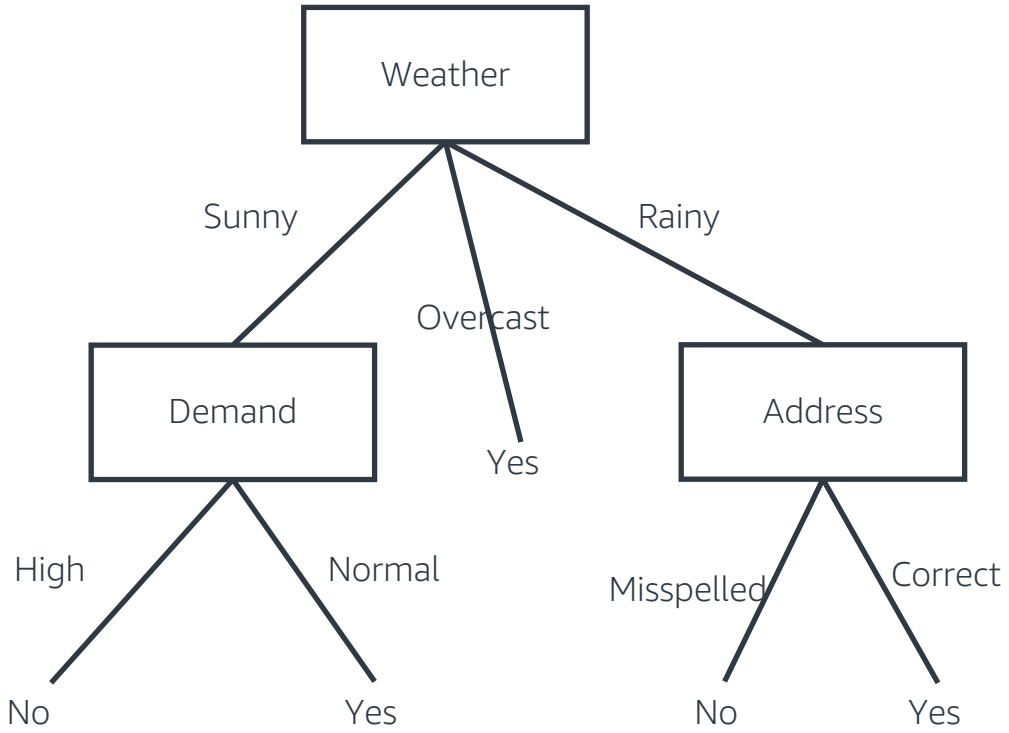
- It has exactly one incoming edge and no outgoing edges.
- It is assigned a class label.

Decision Trees - Example

- Let's predict package delivery timeliness using a decision tree.
- We have the data on the right and tree will be created based on this data.

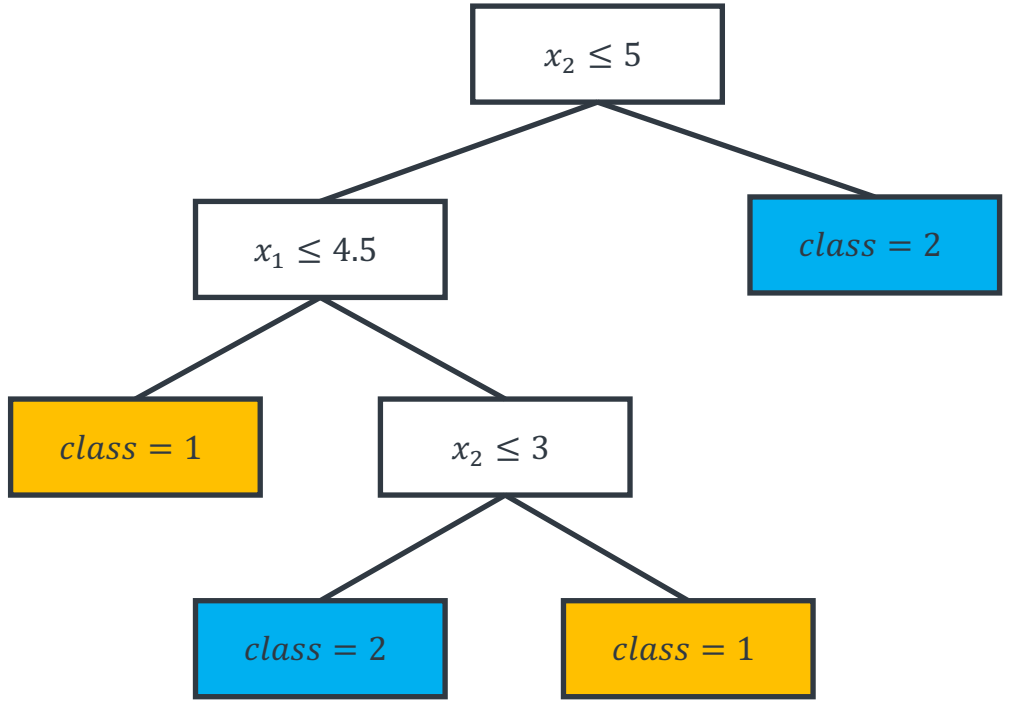
Weather	Demand	Address	ontime
Sunny	High	Correct	No
Sunny	High	Misspelled	No
Overcast	High	Correct	Yes
Rainy	High	Correct	Yes
Rainy	Normal	Correct	Yes
Rainy	Normal	Misspelled	No
Overcast	Normal	Correct	Yes
Sunny	High	Correct	No
Sunny	Normal	Correct	Yes
Rainy	Normal	Misspelled	Yes
Sunny	Normal	Misspelled	Yes
Overcast	High	Misspelled	Yes
Overcast	Normal	Correct	Yes
Rainy	High	Misspelled	No

Decision Trees - Example

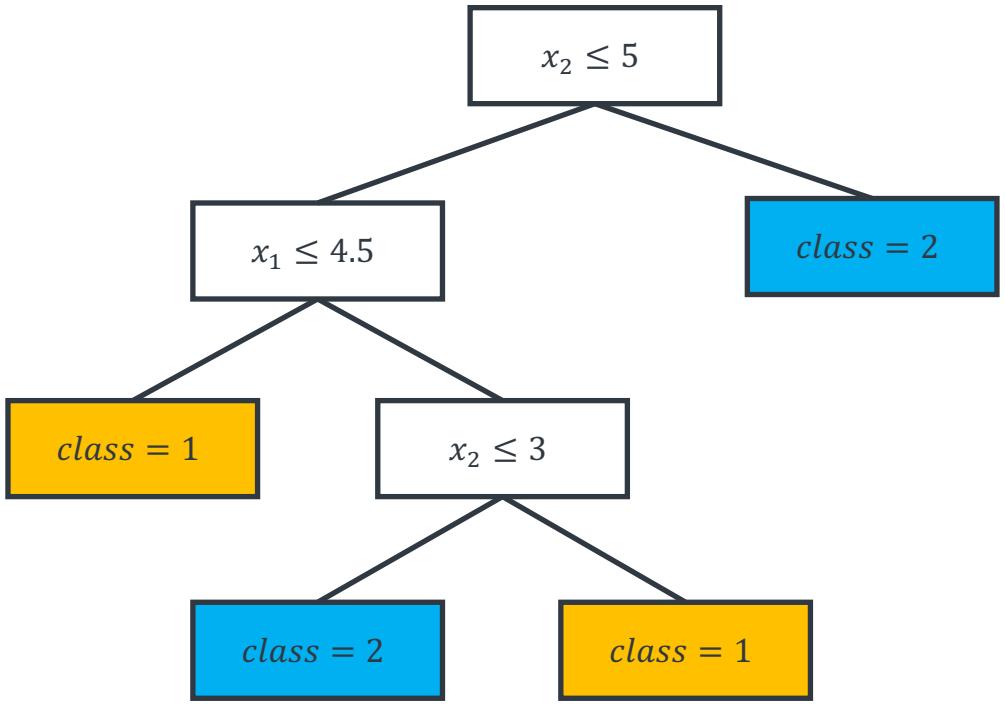


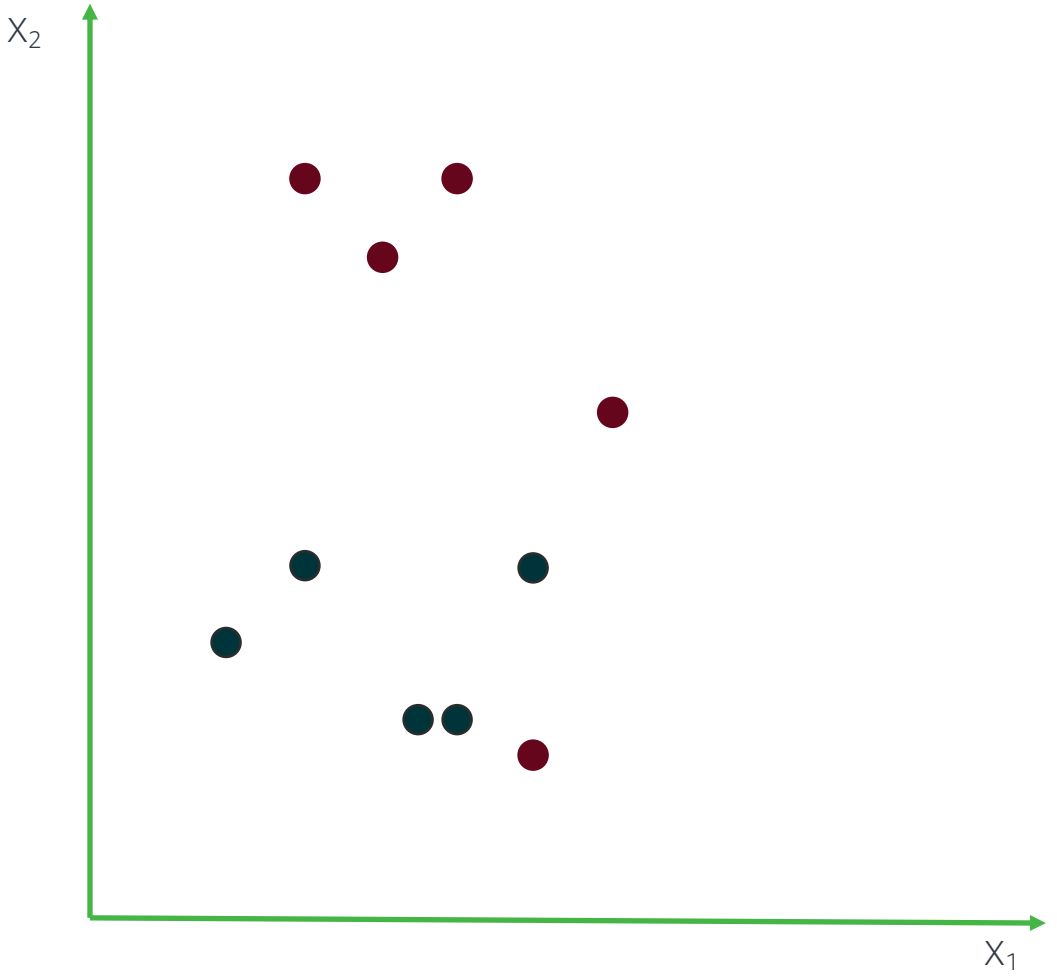
Decision Trees – Numerical example

x_1	x_2	y
3.5	2	1
5	1.5	2
1	3	1
2	4	1
4	2	1
6	6	2
2	9	2
4	9	2
5	4	1
3	8	2

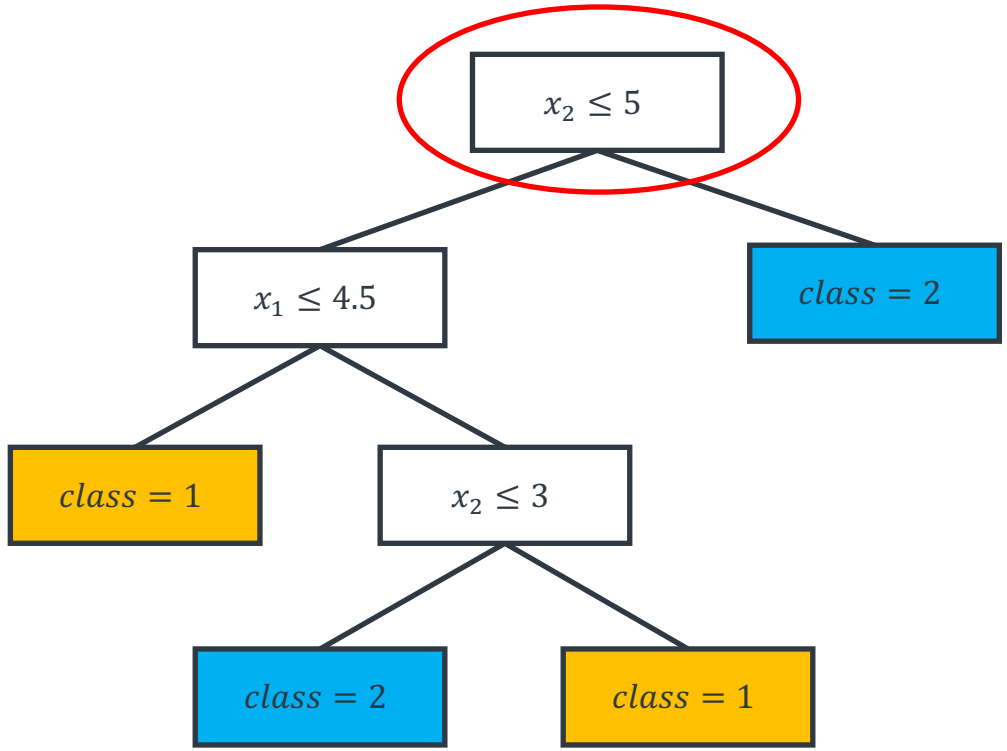


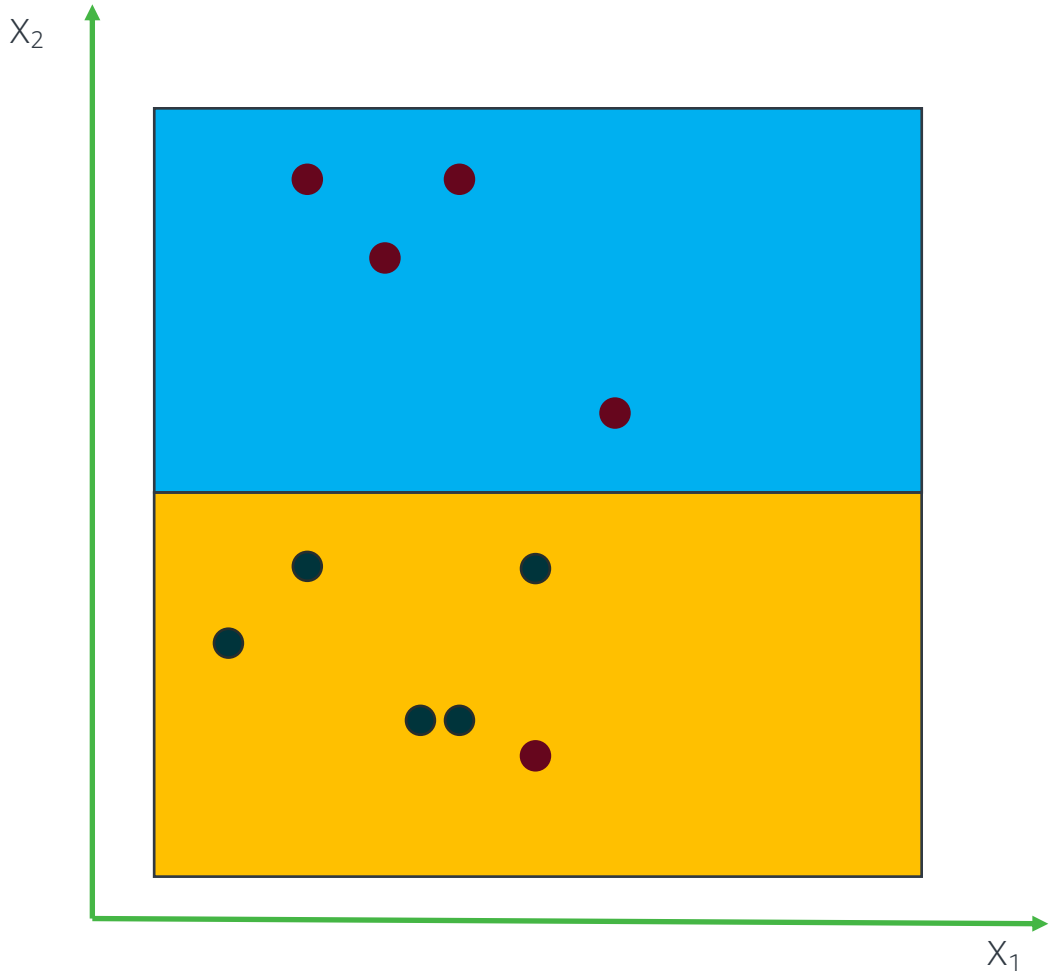
Decision Trees – Numerical example



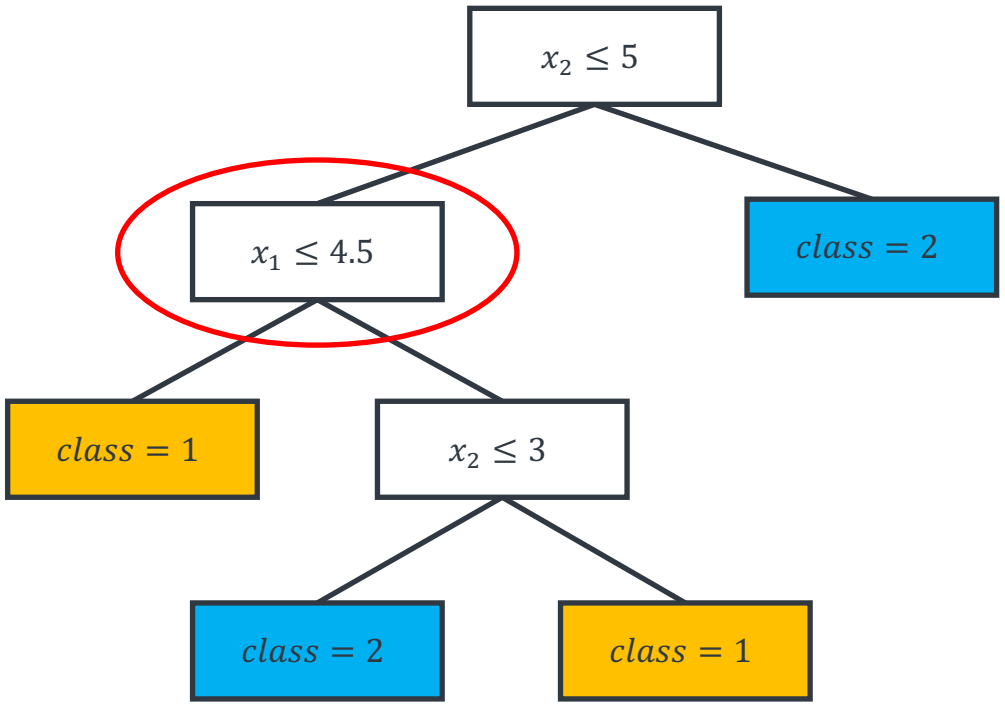


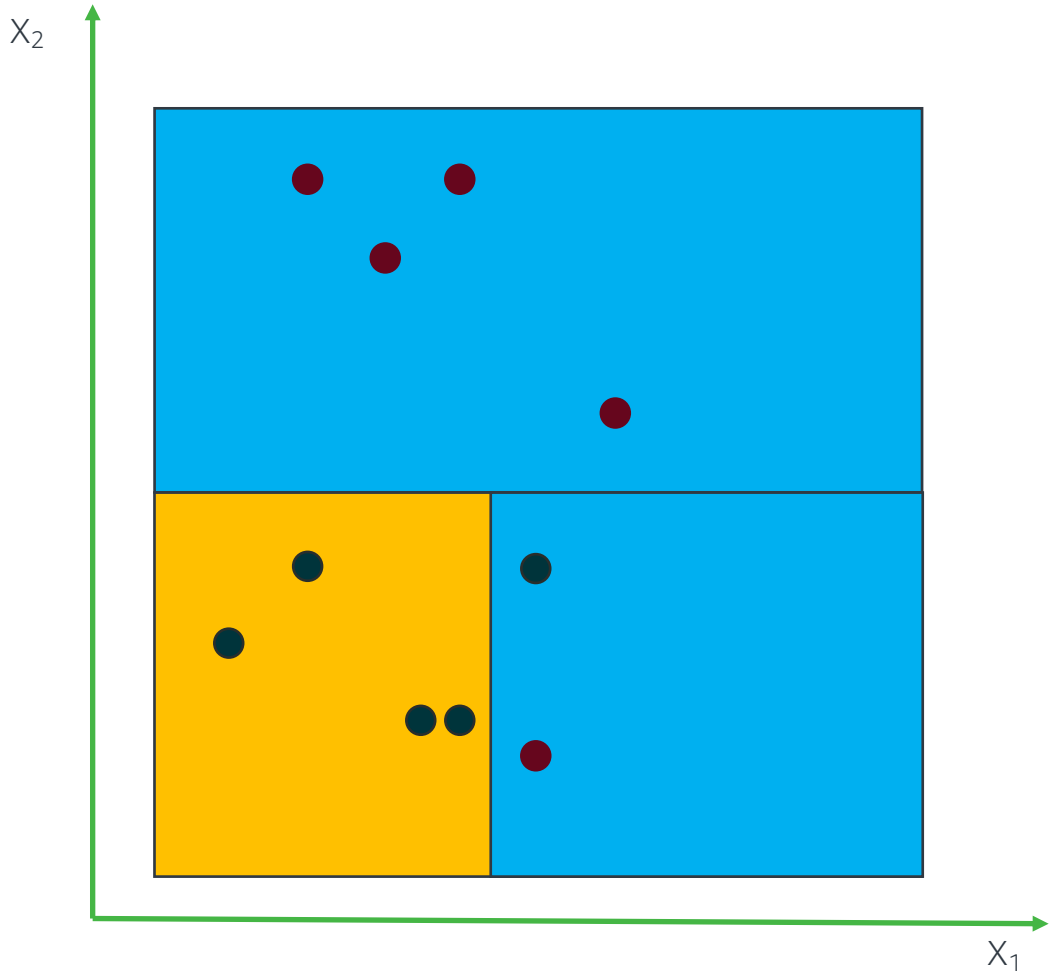
Decision Trees – Numerical example



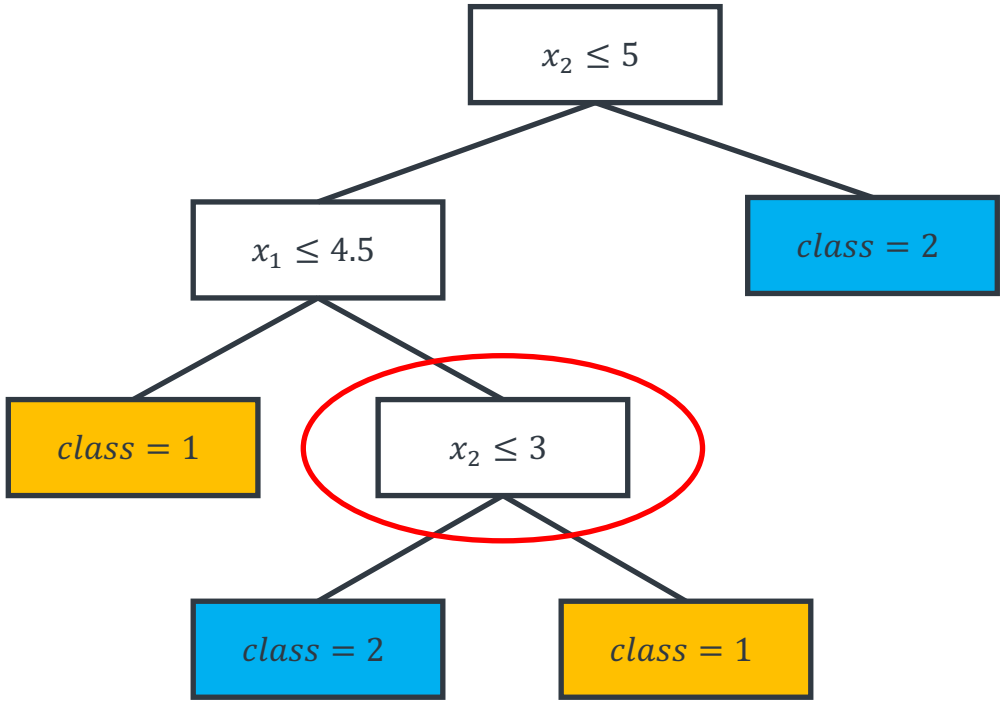


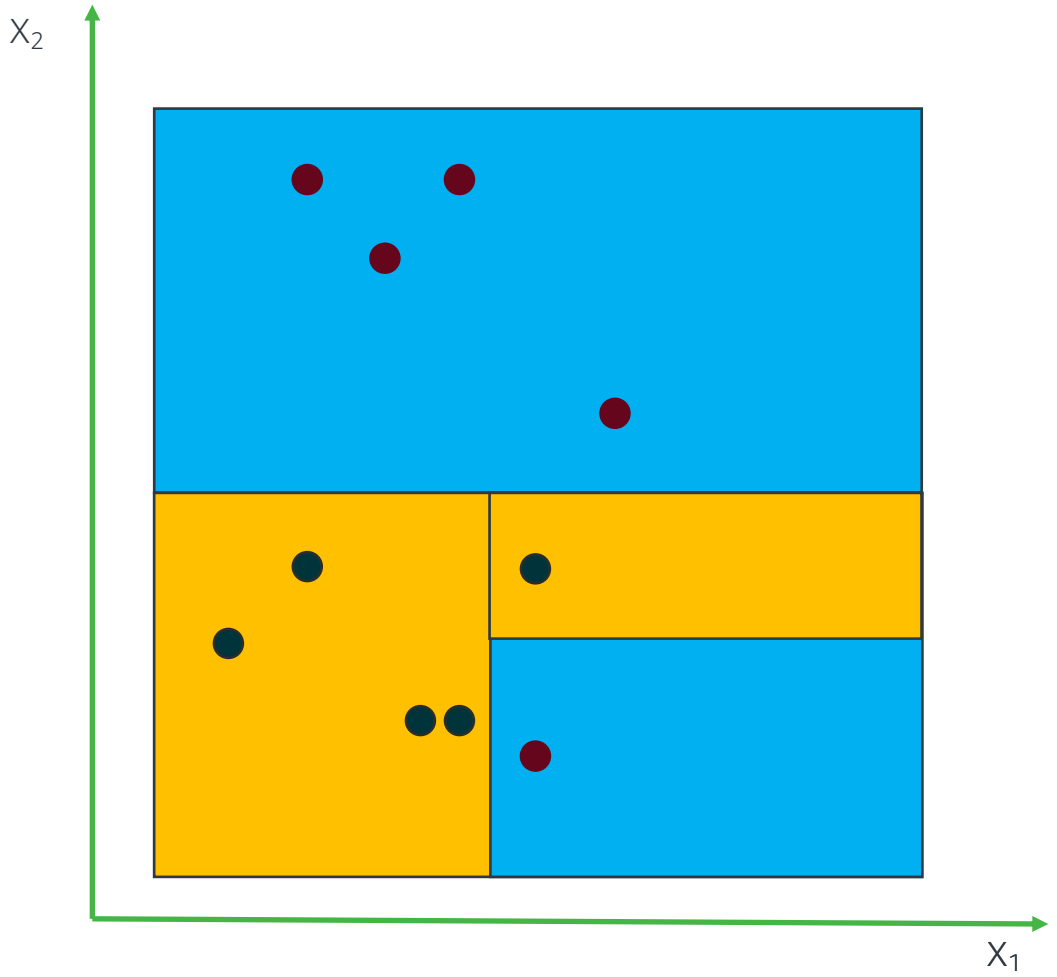
Decision Trees – Numerical example





Encoding with many classes





How to learn a Decision Tree – ID3 Algorithm

Top down approach: Grow the tree from root node to leaf nodes.

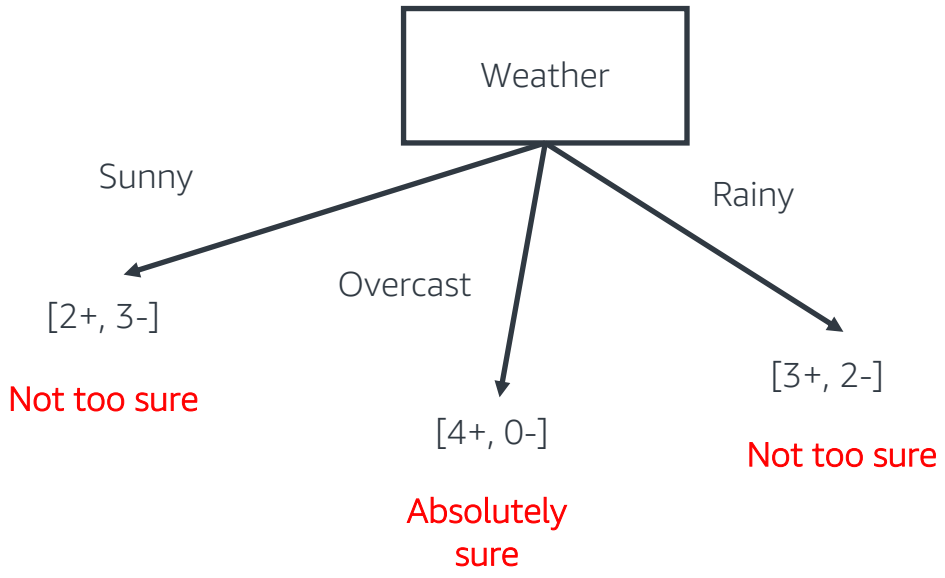
ID3 Algorithm:

Repeat these steps below:

1. Select "the best feature" X_1, X_2, X_3 or X_n (**We will see how to select**)
2. Separate the training samples according to selected feature
3. Stop if we have samples from a single class or if we used all features and note it as a leaf node
4. Assign the leaf node the majority class of the samples in it.

Which feature is the best to split with?

A good split results in overall less uncertainty (impurity). For example:



Weather	Demand	Address	ontime
Sunny	High	Correct	No
Sunny	High	Misspelled	No
Overcast	High	Correct	Yes
Rainy	High	Correct	Yes
Rainy	Normal	Correct	Yes
Rainy	Normal	Misspelled	No
Overcast	Normal	Correct	Yes
Sunny	High	Correct	No
Sunny	Normal	Correct	Yes
Rainy	Normal	Misspelled	Yes
Sunny	Normal	Misspelled	Yes
Overcast	High	Misspelled	Yes
Overcast	Normal	Correct	Yes
Rainy	High	Misspelled	No

How to measure uncertainty

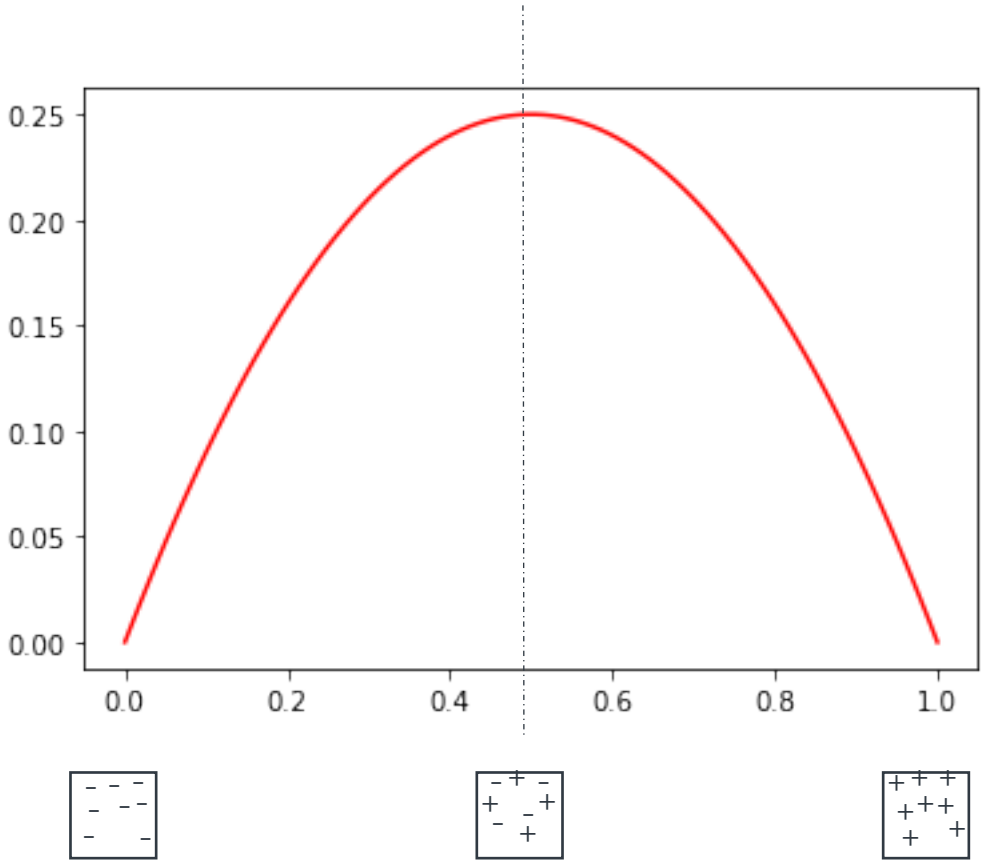
We will use a math term for this

Gini impurity: $i(p_1, \ldots, p_k) = \sum_{k=1}^{n} p_k(1 - p_k)$

n: Number of classes, p_k: prob. of picking a datapoint from class k

Let's look at the graph on the right.

Assume "+" and "–" classes (n=2).



How to measure uncertainty

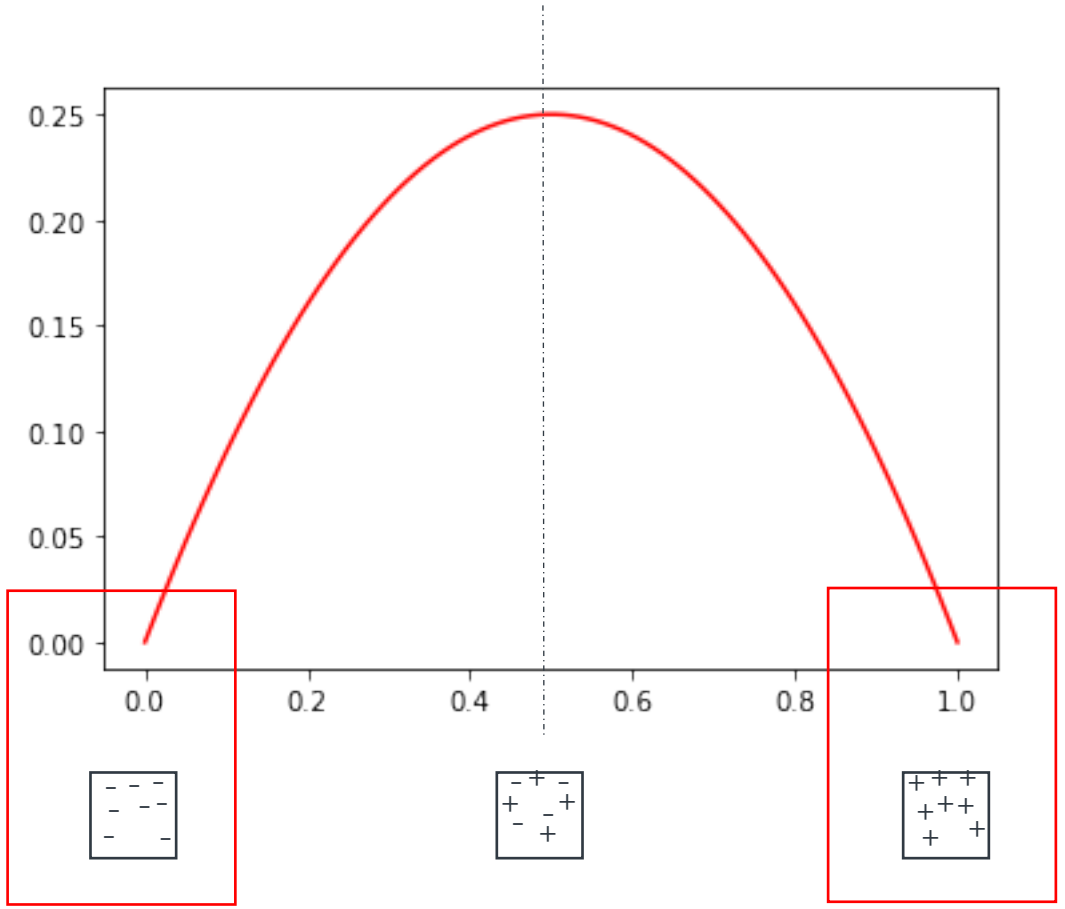
We will use a math term for this

Gini impurity: $i(p_1, \ldots, p_k) = \sum_{k=1}^{n} p_k(1 - p_k)$

n: Number of classes, p_k: prob. of picking a datapoint from class k

Let's look at the graph on the right.

Only + or only – samples have zero impurity



How to measure uncertainty

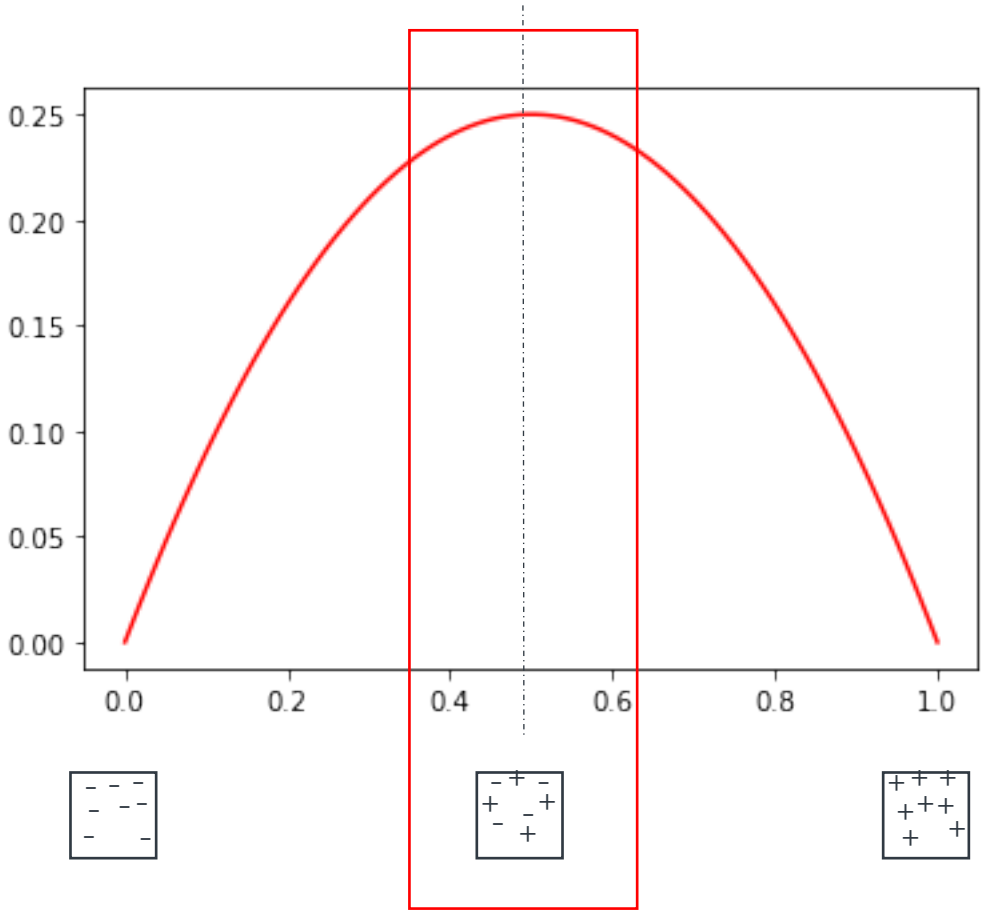
We will use a math term for this

Gini impurity: $i(p_1, \ldots, p_k) = \sum_{k=1}^{n} p_k(1 - p_k)$

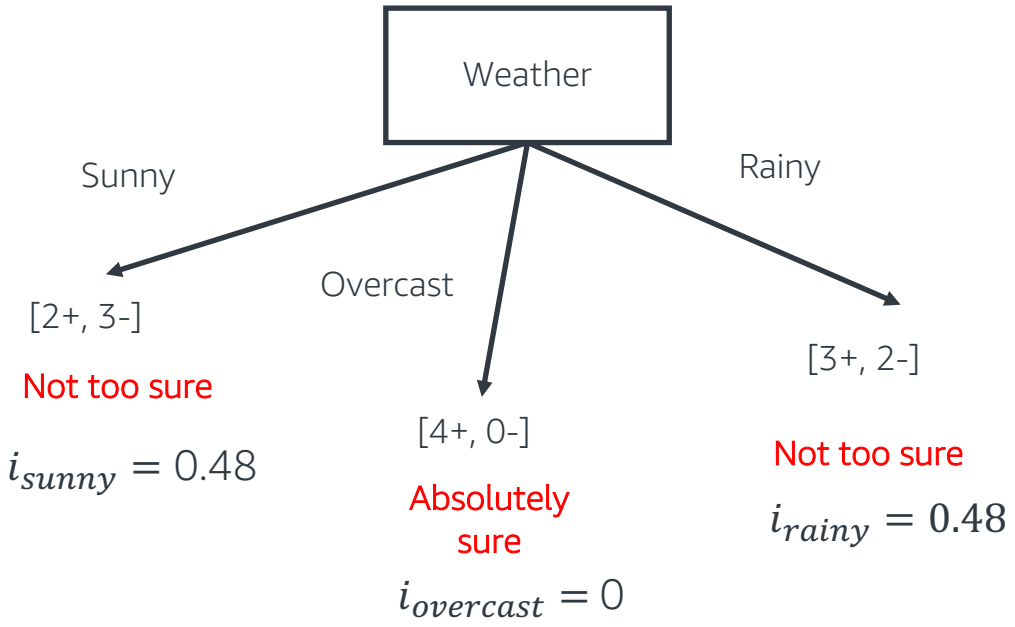
n: Number of classes, p_k: prob. of picking a datapoint from class k

Let's look at the graph on the right.

Mix of + and – samples have high impurity



Calculating gini impurity



Gini impurity: $i(p_1, \ldots, p_k) = \sum_{k=1}^{n} p_k(1 - p_k)$

Example: i_{sunny}= (2/5)*(3/5)+(3/5)*(2/5)
= 0.48

$$i_{weather} = \frac{5}{14} * i_{sunny} + \frac{4}{14} * i_{overcast} + \frac{5}{14} * i_{rainy}$$

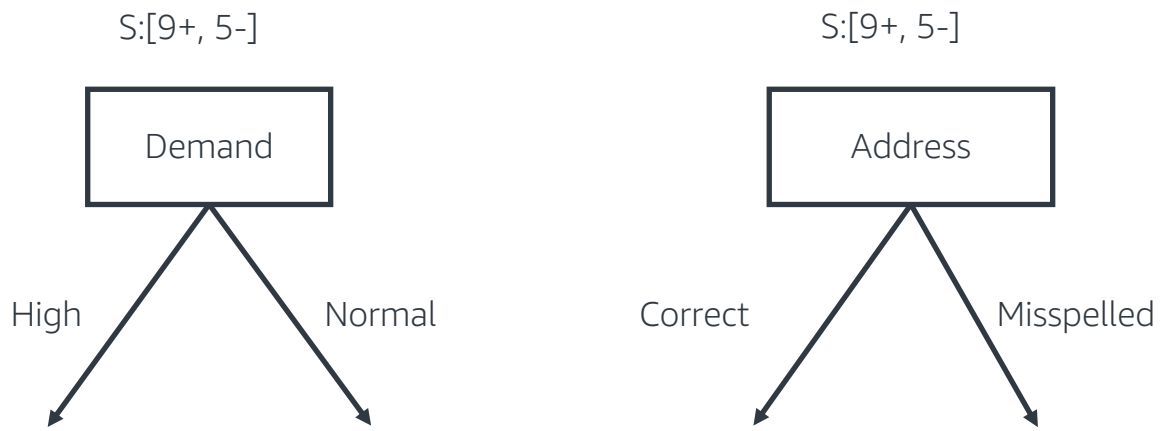
(Weighted sum of impurities)

$i_{weather} = 0.34$

Information Gain and Feature Selection

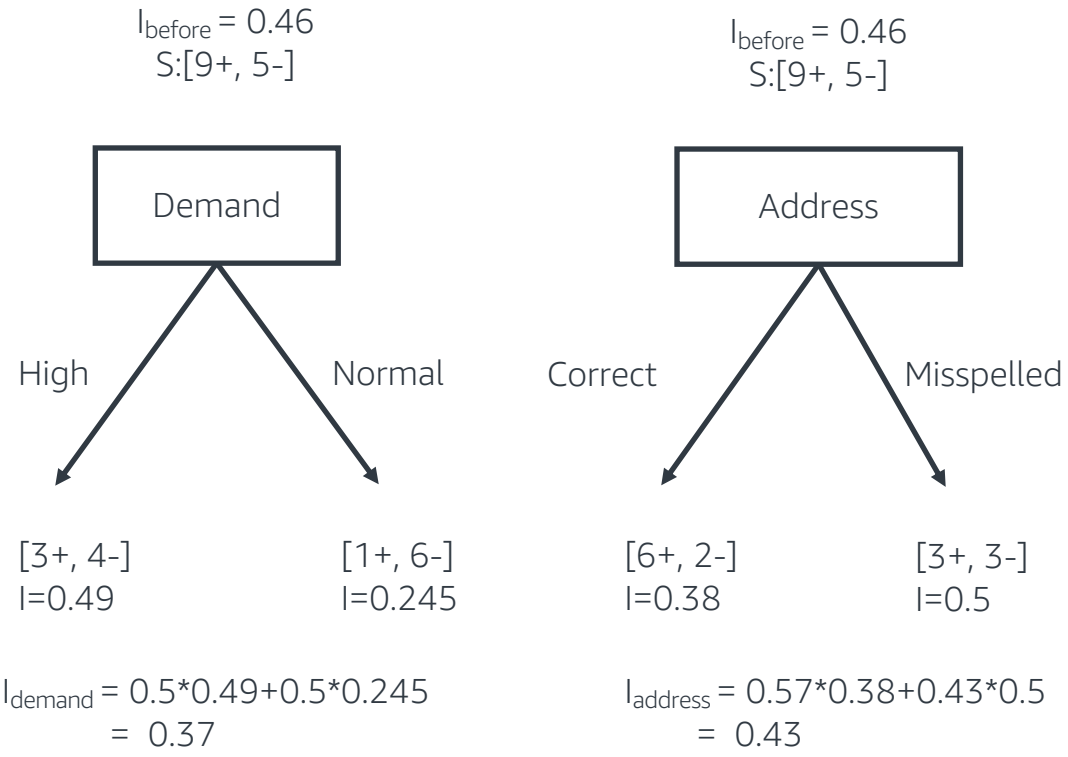
Information gain: Expected reduction in uncertainty due to selected feature.

$$\text{Gain} = \underbrace{i(p_1, p_2, .., p_k)_{before}}_{\text{Impurity before}} - \underbrace{i(p_1, p_2, .., p_k)_{after}}_{\text{Impurity after}}$$



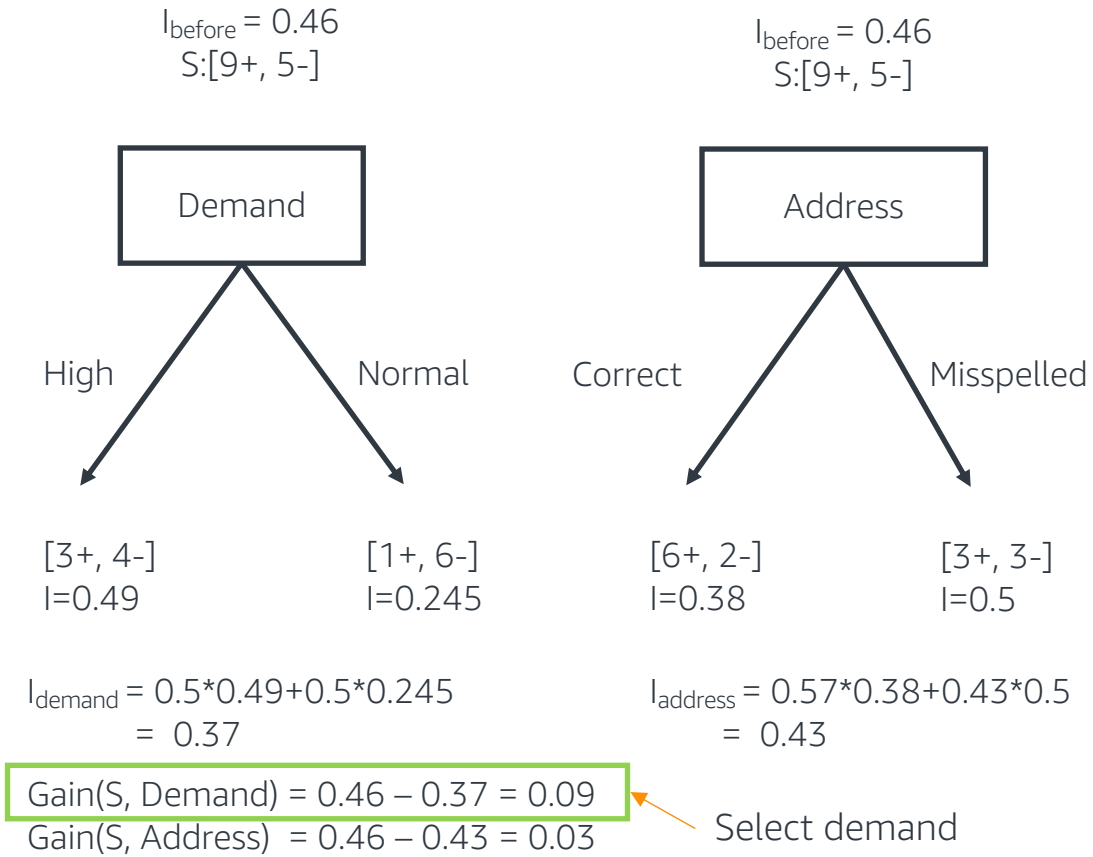
Demand or Address, which one should we select as feature to split?

Information Gain and Feature Selection



Weather	Demand	Address	LateDelivery
Sunny	High	Correct	No
Sunny	High	Misspelled	No
Overcast	High	Correct	Yes
Rainy	High	Correct	Yes
Rainy	Normal	Correct	Yes
Rainy	Normal	Misspelled	No
Overcast	Normal	Correct	Yes
Sunny	High	Correct	No
Sunny	Normal	Correct	Yes
Rainy	Normal	Misspelled	Yes
Sunny	Normal	Misspelled	Yes
Overcast	High	Misspelled	Yes
Overcast	Normal	Correct	Yes
Rainy	High	Misspelled	No

Information Gain and Feature Selection



Weather	Demand	Address	LateDelivery
Sunny	High	Correct	No
Sunny	High	Misspelled	No
Overcast	High	Correct	Yes
Rainy	High	Correct	Yes
Rainy	Normal	Correct	Yes
Rainy	Normal	Misspelled	No
Overcast	Normal	Correct	Yes
Sunny	High	Correct	No
Sunny	Normal	Correct	Yes
Rainy	Normal	Misspelled	Yes
Sunny	Normal	Misspelled	Yes
Overcast	High	Misspelled	Yes
Overcast	Normal	Correct	Yes
Rainy	High	Misspelled	No

Information Gain and Feature Selection

Let's compare gains for each attribute

Gain(S, Weather) = 0.46-0.34=0.12

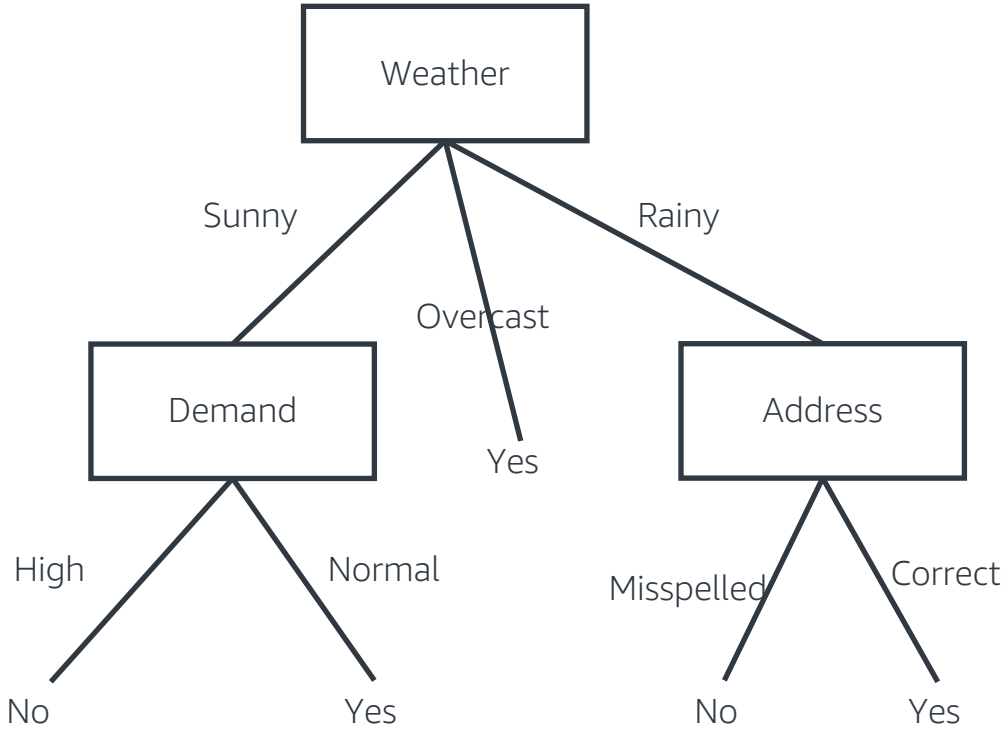
Gain(S, Demand) = 0.46-0.37=0.09

Gain(S, Address) = 0.46-0.43=0.03

Weather is the highest of all, we will start the tree with Weather feature as the root node.

Weather	Demand	Address	LateDelivery
Sunny	High	Correct	No
Sunny	High	Misspelled	No
Overcast	High	Correct	Yes
Rainy	High	Correct	Yes
Rainy	Normal	Correct	Yes
Rainy	Normal	Misspelled	No
Overcast	Normal	Correct	Yes
Sunny	High	Correct	No
Sunny	Normal	Correct	Yes
Rainy	Normal	Misspelled	Yes
Sunny	Normal	Misspelled	Yes
Overcast	High	Misspelled	Yes
Overcast	Normal	Correct	Yes
Rainy	High	Misspelled	No

Recap ID3 Algorithm



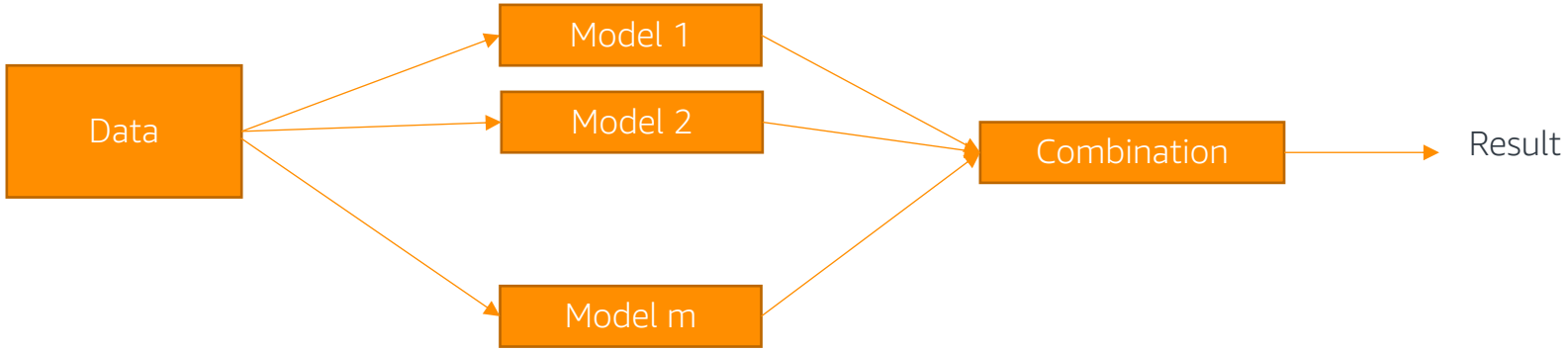
ID3 Algorithm:

Repeat these steps below:

1. Select "the best feature" X_1, X_2, X_3 or X_n by considering the **Information Gain (max decrease in impurity)** of each feature
2. Separate the training samples according to selected feature
3. Stop if we have samples from a single class or if we used all features and note it as a leaf node
4. Assign the leaf node the majority class of the samples in it.

Ensemble Learning

Ensemble models create a strong model from multiple weak models.



In this section, we will cover **Bootstrap Aggregating (Bagging)** and **Boosting** techniques.

Bootstrap Aggregating (Bagging)

- In ensemble learning, we will train multiple models. We need to provide data to each of the individual model.
- With **Bootstrap technique**, we can draw synthetic data from the original dataset. We do this by randomly sampling with replacement.
- For example: Given dataset: `[1, 2, 4, 5, 7, 9,10]`
- Potential samples:
  - `[ 1, 1, 2, 4, 9,10,10]`
  - `[ 2, 4, 5, 5, 7, 7, 7]`
  - `[ 1, 1, 1, 1, 1, 1, 1]`
  - `[ 1, 2, 4, 5, 7, 9,10]`
  - ….

Bootstrap Aggregating (Bagging)

- Applying the bootstrapping technique:
  - Given a dataset D with n samples in it:
  - Separate it into m different random samples with replacement: D_m
    - These may have a smaller number of points if desired
  - Train m models f_m on the sampled datasets D_m
  - Produce the final prediction by either averaging f_m (for regression problems) or voting with f_m (for classification problems)

Bagging in Sklearn

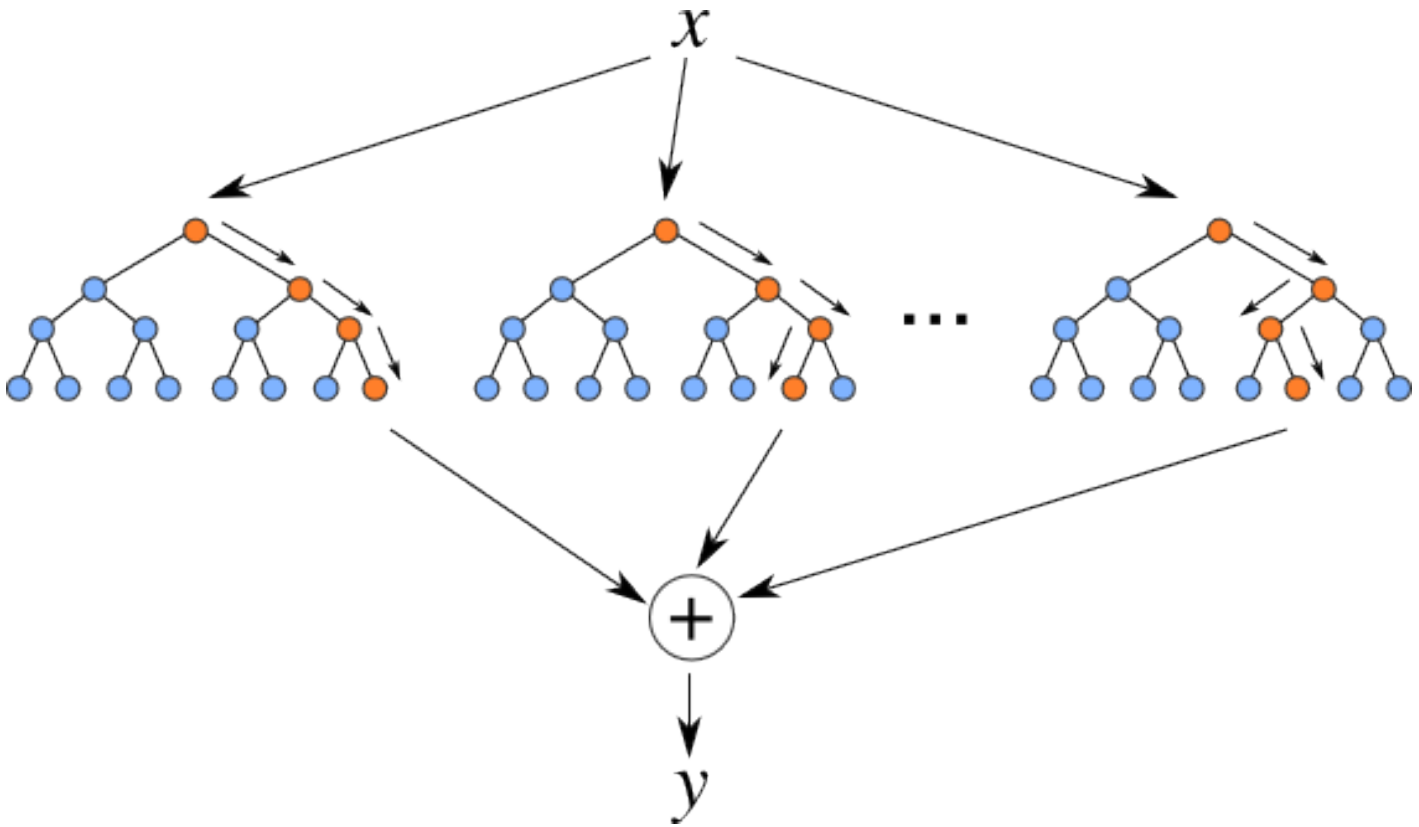
- `Sklearn` provides a very general interface for bagging which can be provided any `base_estimator`:

```
BaggingClassifier(base_estimator=None, n_estimators=10, max_samples=1.0, bootstrap=True, oob_score=False, n_jobs=1)
```

- The full interface is larger.

Random Forests

Combination of multiple trees.



Train a decision tree for each sampled data.

Combine end results of each tree by voting.

Topics for today

- Data Imputation and Scaling
- Feature Engineering
- Trees and Ensemble Learning
- Hyperparameter tuning
- Sagemaker
- Final Project

Hyperparameter Tuning

Hyperparameters are parameters that affect the performance and structure of an ML algorithm.

With hyperparameter tuning, we look for the best combination of hyperparameters in a ML model.

For example:

- **Decision trees:** Depth, max number of splits, impurity measure (entropy, gini etc.)
- **K Nearest Neighbors**: K (number of neighbors)
- **Ensemble Bagging model:** Number of individual models

Grid Search

- **Grid search** is the most basic hyperparameter tuning method.
- It is an exhaustive search to find the optimum combination of hyper parameters.
- We provide a list parameters to test against an unknown dataset.
- For Decision tree below:

```
{
    "depth": [5, 10, 50, 100, 250],
    "max splits": [15, 20, 25],
    "impurity measure": ["gini", "entropy"]
}
```

Total hyperparamers: 5 x 3 x 2 = 30

[5, 15, "gini"]

[10, 15, "gini"]

....

....

Random Search

- Instead of trying every possible combination, **Random Search** chooses a random set of values and only try those.
- Random search can use distributions to sample hyperparameters from.
- As described in the original paper: Random Search for Hyper-Parameter Optimization, J. Bergstra & Y. Bengio (2012)
  - "Grid search spends too much time evaluating unpromising regions of the hyperparameter search space because it has to evaluate every single combination in the grid"

Bayesian Search

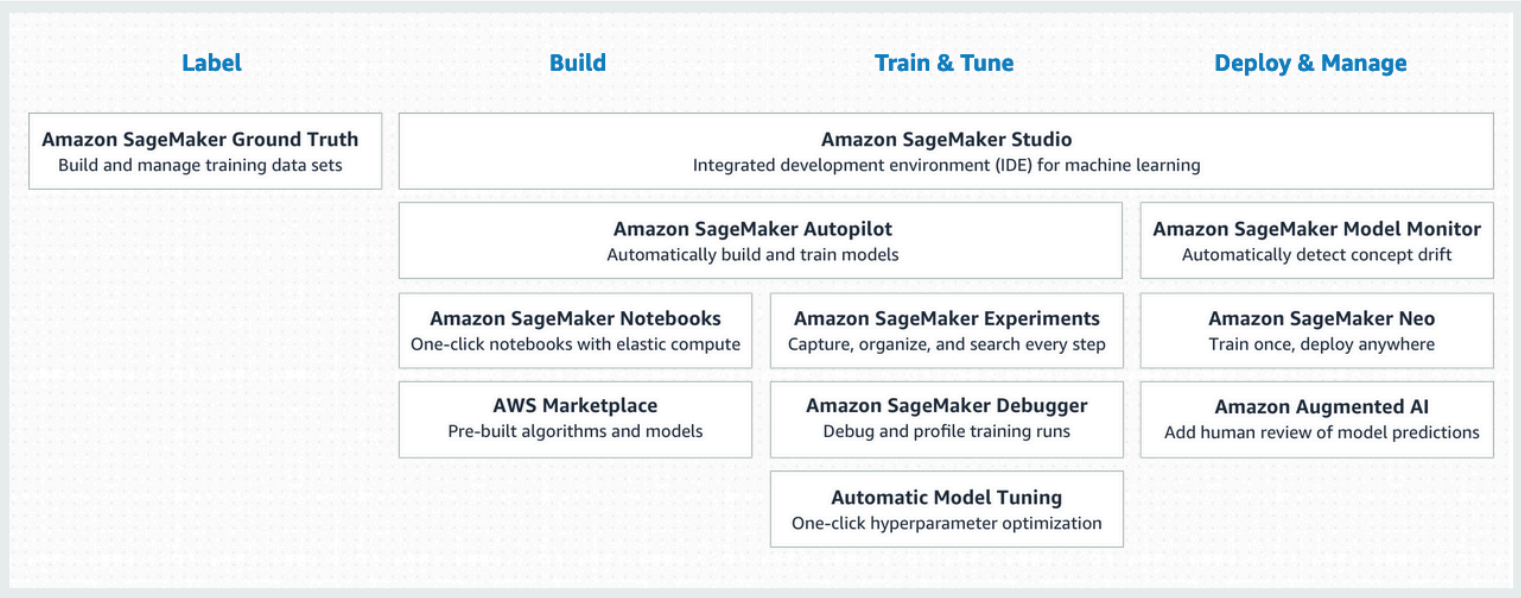
- Bayesian Search method keeps track of previous hyperparameter evaluations and builds a probabilistic model.
- It tries to balance exploration (uncertain hyperparameter set) and exploitation (hyperparameters with a good chance of being optimum)
- It prefers points near the ones that worked well
- Sagemaker uses Bayesian Search for hyperparameter optimization.

Hands-on exercise

- In this exercise we will work with an internal Amazon dataset.
- We will do the following tasks:
  - Exploratory Data Analysis
  - Splitting dataset into training and test sets.
  - Fit a Random Forest Classifier.
  - Check the performance metrics on test set.
- Notebook link: https://eider.corp.amazon.com/sazaracs/notebook/NBV03AKTMEHX

Sagemaker – Train and deploy model

- AWS service to easily train, test and deploy ML models: https://aws.amazon.com/sagemaker/



- We will train and deploy a simple ML model. Files are uploaded here.

Data Labeling – Sagemaker Groundtruth

- Machine learning can be applied in many different areas. With this, we usually have many different types of labels.
- We will use Sagemaker Groundtruth tool and label some sample data.
- Groundtruth allows users to create labeling tasks and assign them to internal team members or outsource them.
- Demo files are available here.

Text Tasks

Task category

Select the type of data being labeled to view available task templates for it or select 'Custom' to create your own.

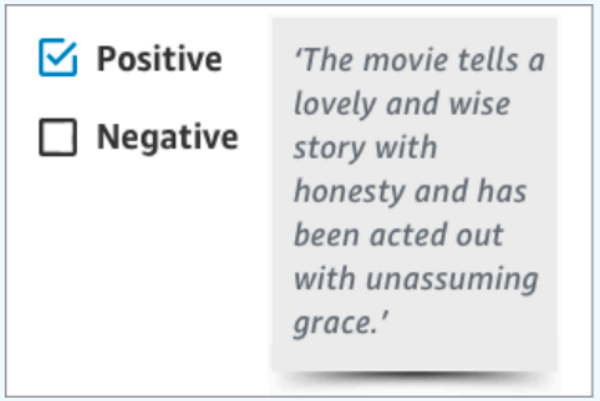
Text

Task selection

Select the task that a human worker will perform to label objects in your dataset.

Text classification

Get workers to categorize text into specific classes. Info



Named entity recognition

Get workers to apply labels to words or phrases within a larger text. Info

Image Tasks

Task selection

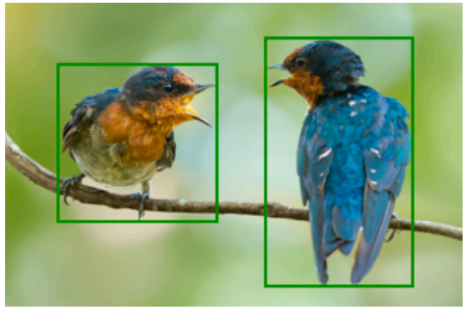
Select the task that a human worker will perform to label objects in your dataset.

Image classification

Get workers to categorize images into specific classes. **Info**

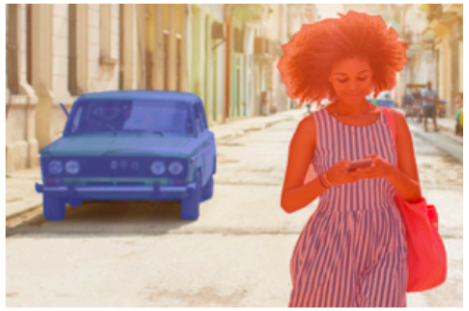
Bounding box

Get workers to draw bounding boxes around specified objects in your images. **Info**



Semantic segmentation

Get workers to draw pixel level labels around specific objects and segments in your images. **Info**

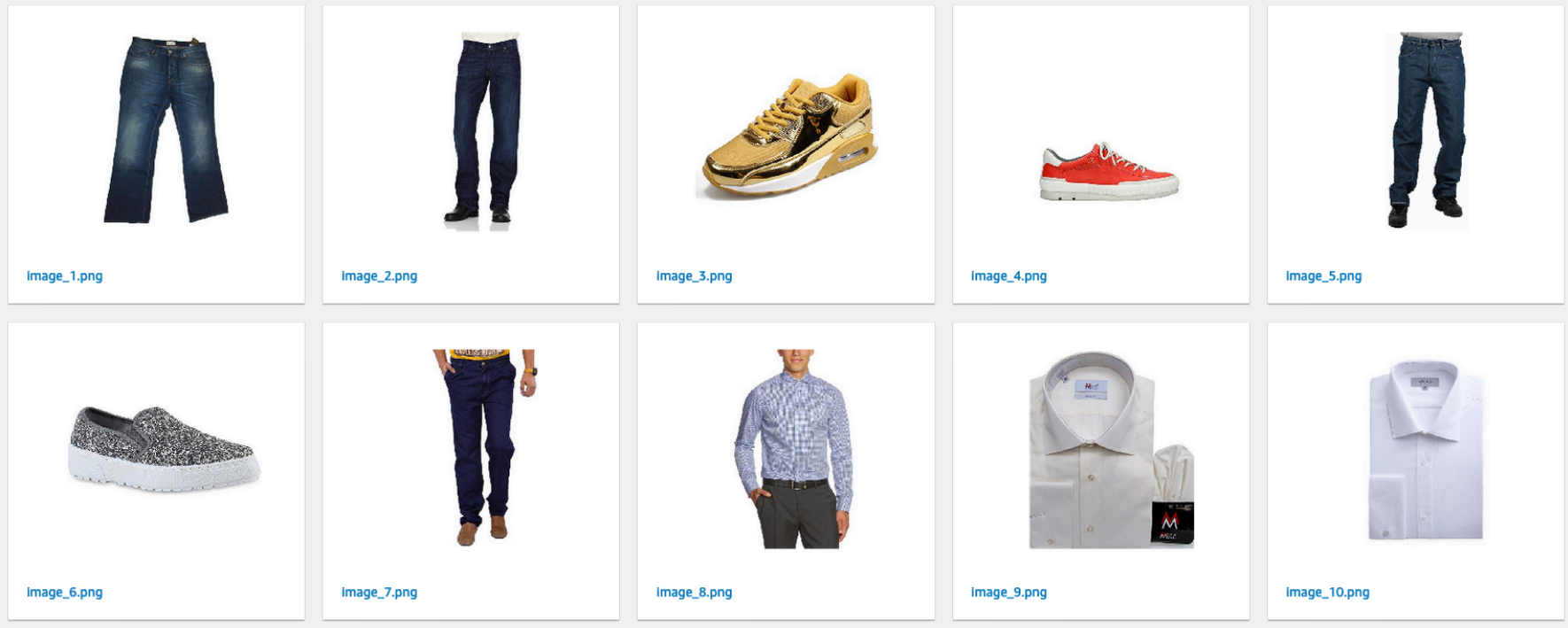


Label verification

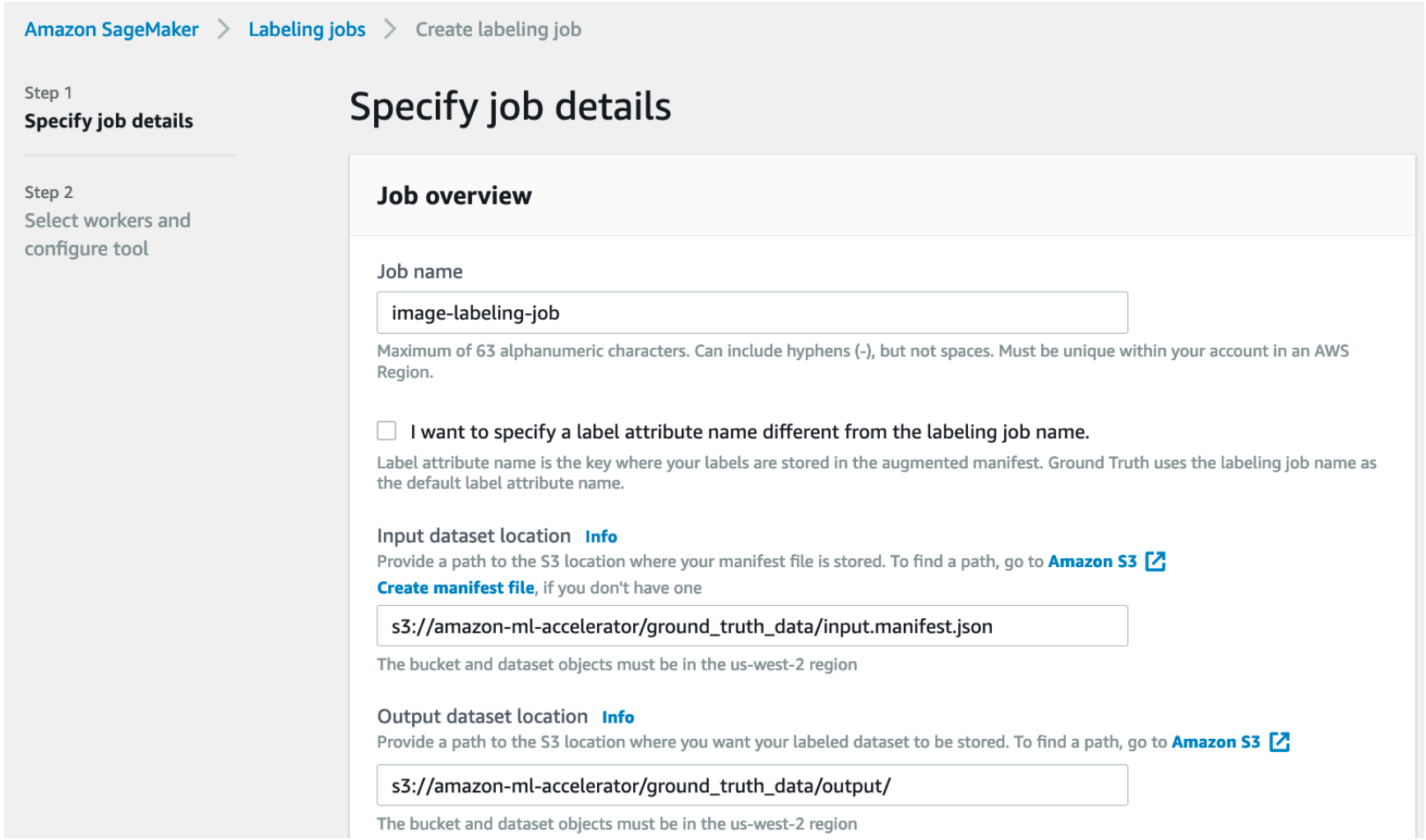
Get workers to verify existing labels in your dataset. **Info**

Groundtruth Demo

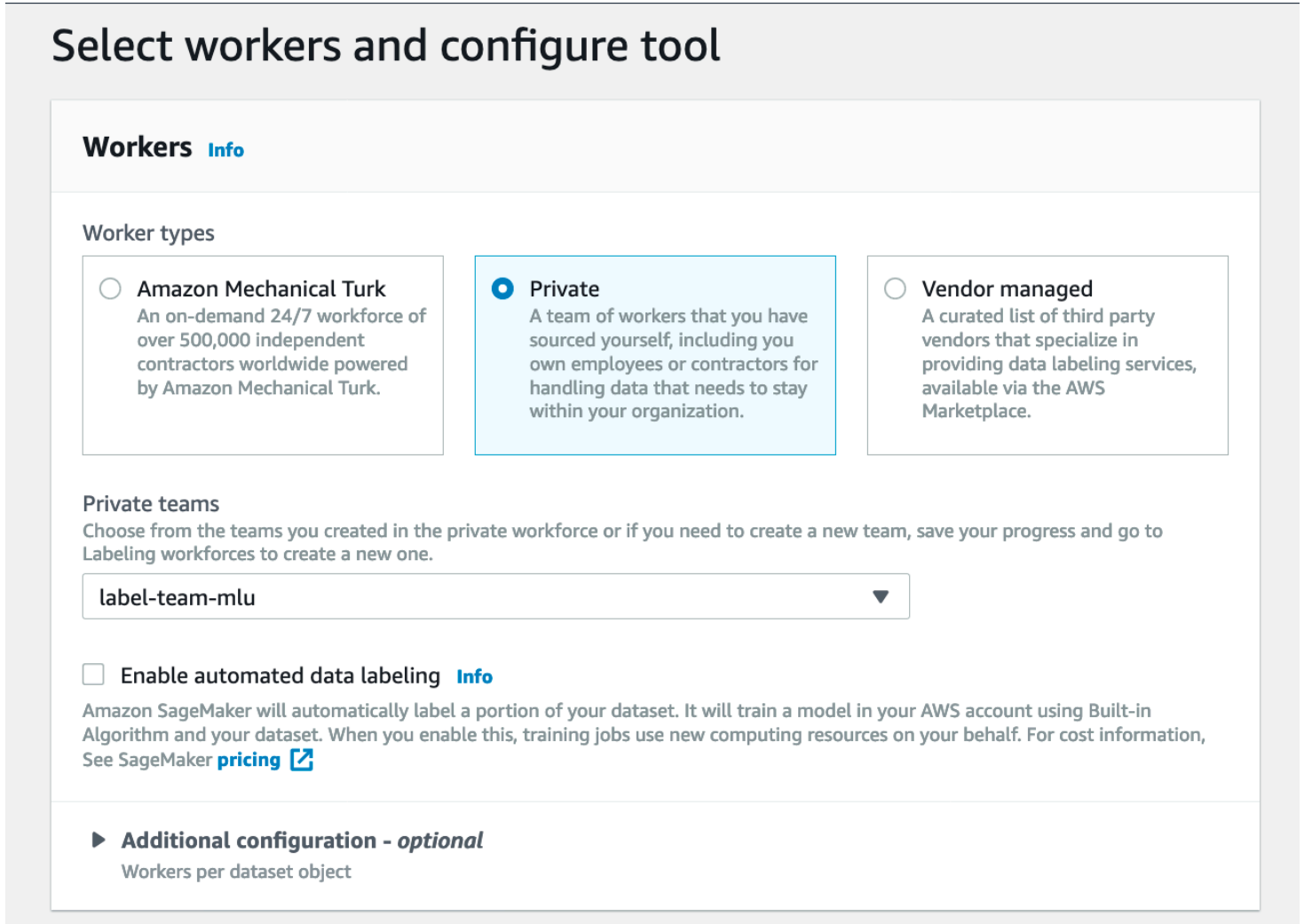
Assume we have these 10 images to label:



Groundtruth Demo



Groundtruth Demo



You can checkout this video walkthrough more details:

https://youtu.be/8J7y513oSsE

Topics for today

- Data Imputation and Scaling
- Feature Engineering
- Trees and Ensemble Learning
- Hyperparameter tuning
- Sagemaker
- Final Project

Final Project

Product substitute: Given products (A, B), predict whether B is a substitute for A

- We say that B is a "substitute" for A if a customer would buy B in place of A -- say, if A were out of stock.
- **The goal** of this project is to predict a substitute relationship between pairs of products.
- **Link:** https://leaderboard.corp.amazon.com/tasks/478

Final Project Walkthrough – Day 2

- See the provided project walkthrough below.
- Use categorical variables and decision trees/random forests

- Complete the following notebook and submit your results

https://eider.corp.amazon.com/sazaracs/notebook/NBAJPGY1ZTDZ

Notebooks for day 2

- Text processing example: https://eider.corp.amazon.com/sazaracs/notebook/NBQM6DD1RI0P
- Categorical variables and grid search: https://eider.corp.amazon.com/sazaracs/notebook/NBV03AKTMEHX
- Final project walkthrough for day 2: https://eider.corp.amazon.com/sazaracs/notebook/NBAJPGY1ZTDZ