Course Overview

Lecture 1

- Introduction to ML
- Model Evaluation
  - Train-Validation-Test
  - Overfitting
- Exploratory Data Analysis

Lecture 2

- Feature Engineering
- Tree-based Models
  - Decision Tree
  - Random Forest
- Hyperparameter Tuning

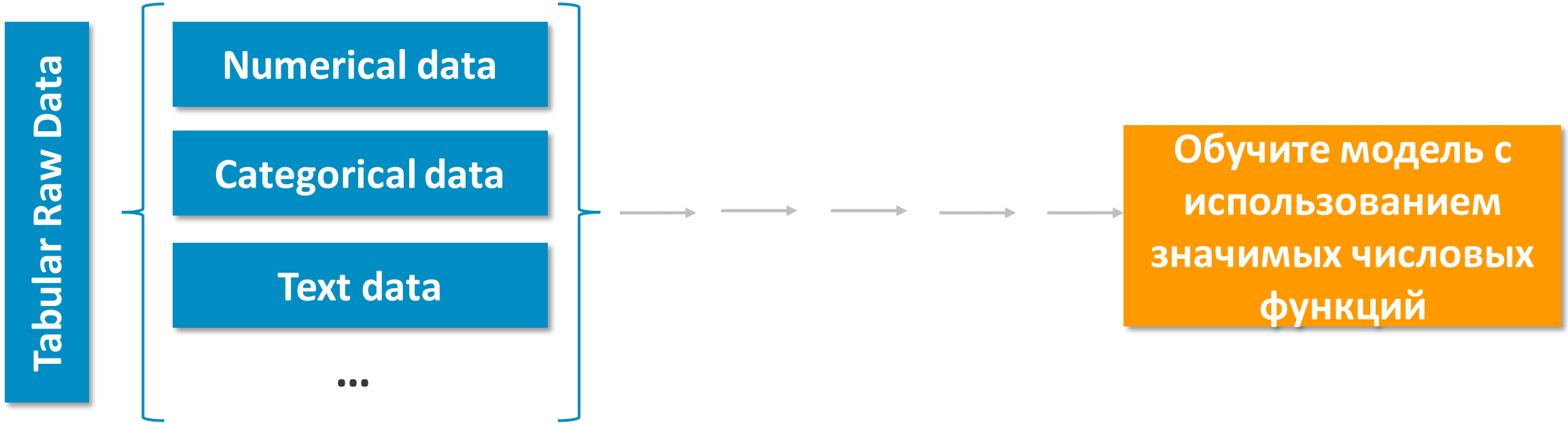
Lecture 3

- Optimization
- Regression Models
- Regularization
- Boosting
- Neural Networks
- AutoML

Feature Engineering

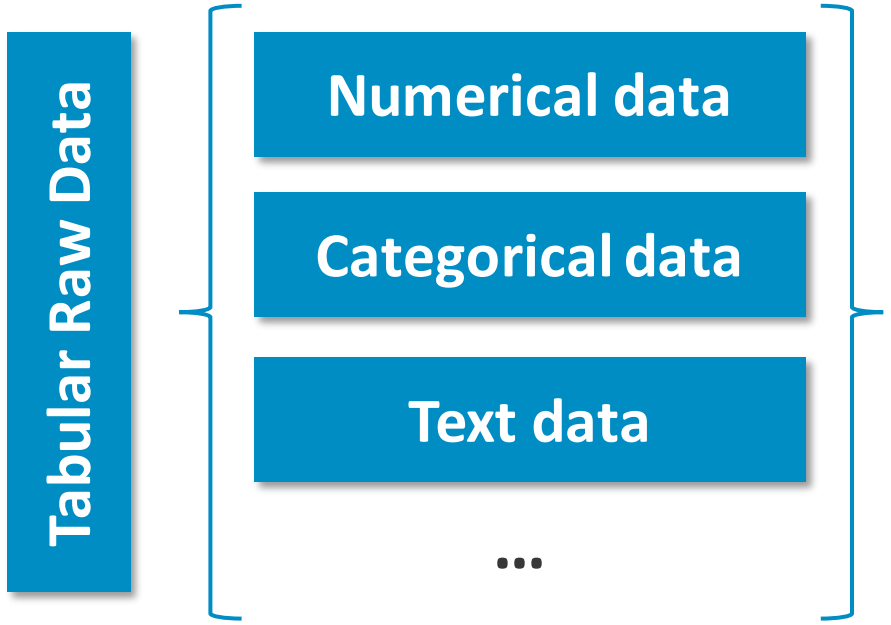
Feature Engineering

- **Feature engineering**: Используйте **знания предметной области** и данных для создания **новых** функций в качестве входных данных для моделей машинного обучения на основе предоставленных **необработанных данных**.



Feature Engineering

- **Feature engineering**: Используйте знания **предметной области** и данных для создания **новых** функций в качестве входных данных для моделей ML на основе предоставленных необработанных данных.



Интуиция: какую информацию использовал бы человек для предсказания? Часто больше искусства, чем науки!

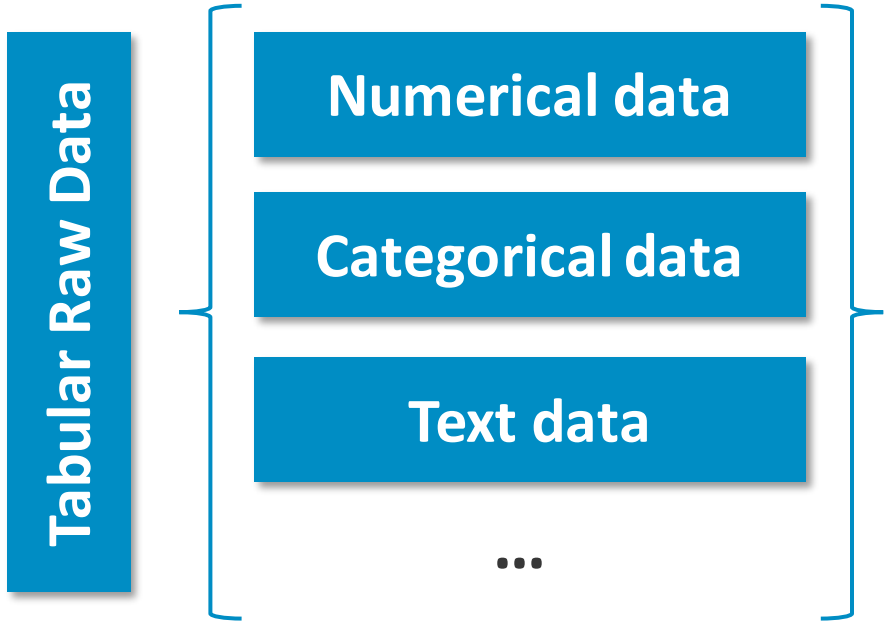
Feature Engineering

- **Select** features

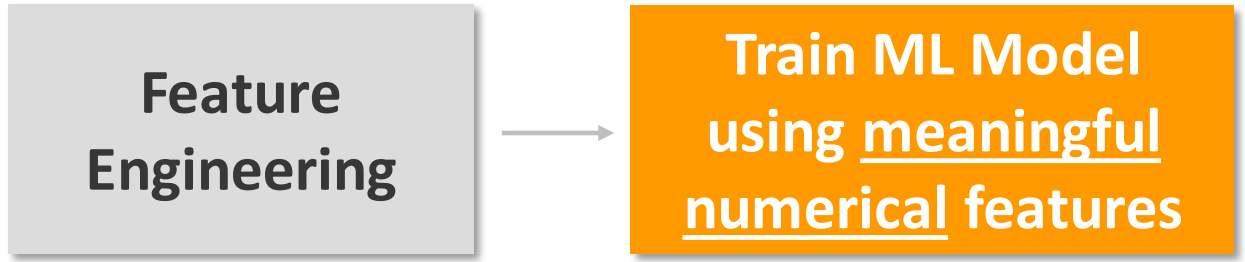
Обучите модель с использованием значимых числовых функций

Feature Engineering

- **Feature engineering**: Используйте знания **предметной области** и данных для создания **новых** функций в качестве входных данных для моделей ML на основе предоставленных необработанных данных.



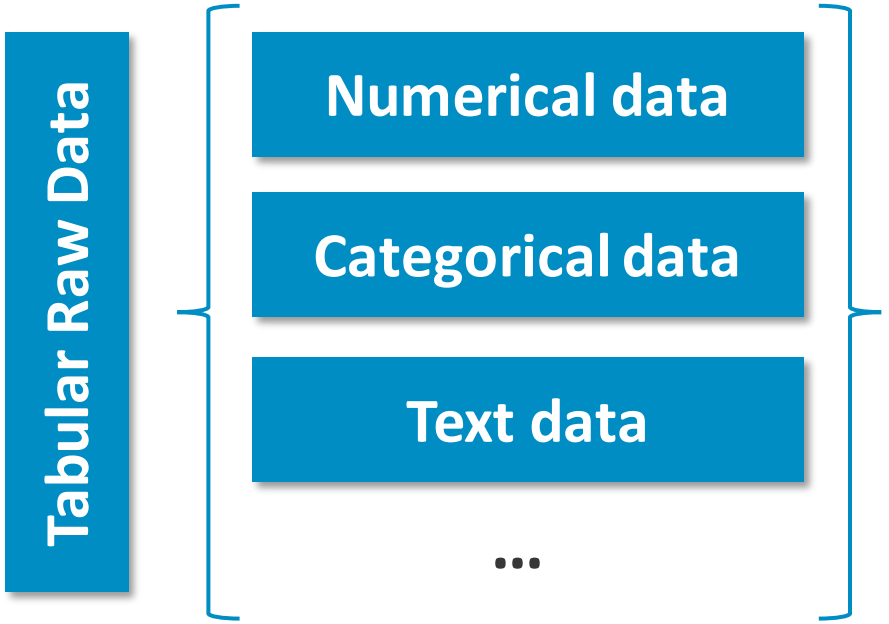
Интуиция: какую информацию использовал бы человек для предсказания? Часто больше искусства, чем науки!



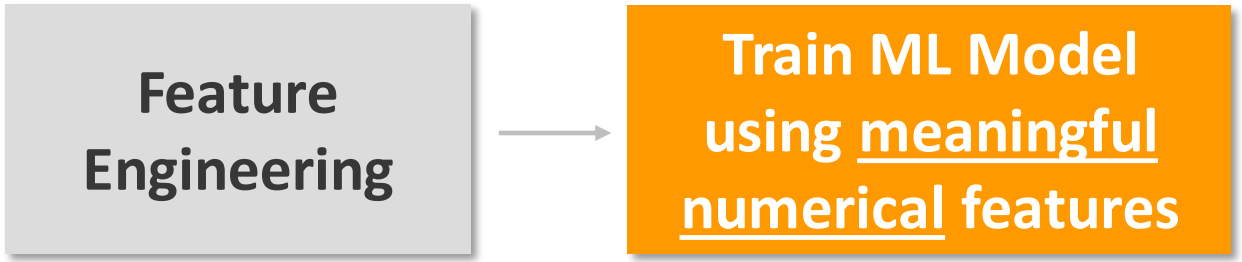
- **Select** features
- Построение признаков (Feature construction): умножение, возведение в квадрат, полиномиальные функции, log(), другие ядра и др.

Feature Engineering

- **Feature engineering**: Используйте знания предметной области и данных для создания новых функций в качестве входных данных для моделей ML на основе предоставленных необработанных данных.



Интуиция: какую информацию использовал бы человек для предсказания? Часто больше искусства, чем науки!



- **Select** features
- Feature **construction** (multiplication, squaring, etc)
- Извлечение признаков (Feature **extraction):** encoding, vectorization
- Выбор признаков (Feature **selection):** dimensionality reduction

Encoding Categoricals

Encoding Categorical Features

- **Categorical** (также называется дискретным) **features**: Эти функции не имеют естественного числового представления.
  - **Example**: color ∈ {green, red, blue}, это мошенничество ∈ {false, true}
  - Большинство моделей машинного обучения требуют преобразования категориальных признаков в числовые.
- **Encode/define a mapping**: Присвойте номер каждой категории.
  - **Ordinals**: Категории упорядочены, например, size ∈ {L > M > S}. Мы можем назначить L->3, M->2, S->1.
  - **Nominals**: Категории неупорядочены, например, color. Мы можем присвоить номера случайным образом.

Encoding Categorical Features

- **LabelEncoder: sklearn** encoder, кодирует целевые метки со значением от 0 до n_classes-1 .fit(), .transform()
  - Кодирует значения целевых меток, y (или **только одна функция!**), а не вход X.
  - Can be used to transform non-numerical labels or numerical labels.

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

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['color'] = le.fit_transform(df['color'])
```

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

Давайте закодируем одну функцию, например поле color.

Encoding Categorical Features

- **OrdinalEncoder: sklearn** encoder, кодировать категориальные признаки как целочисленный массив .fit(), .transform()
  - Encodes (**два или более**) categorical features (**не работает с одной функцией**!)
  - Возвращает один столбец целых чисел (от 0 до n_categories - 1) для каждой функции.

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

```
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder()

df[['color','size','classlabel']] =
oe.fit_transform(df[['color','size','classlabel']])
```

	color	size	price	classlabel
0	1.0	2.0	10.1	1.0
1	2.0	1.0	13.5	0.0
2	0.0	0.0	15.3	1.0

Закодируем все категориальные поля.

Encoding Categorical Features

Problem: Кодирование категориальных функций целыми числами неверно, потому что порядок и размер целых чисел бессмысленно.

One-hot-encoding: Разделите категориальные функции на множество двоичных функций (столько категорий для каждой функции).

- **OneHotEncoder: sklearn** one-hot encoder, кодирует категориальные функции как one-hot числовой массив .fit(), .transform()
  - создает двоичный столбец для каждой категории и возвращает разреженную матрицу или плотный массив.
  - Работает с двумя или более функциями (для быстрого кодирования только одной функции следует использовать LabelBinarizer!)
  - **get_dummies: pandas** one-hot encoder

Encoding Categorical Features

get_dummies: **pandas** one-hot encoder, преобразует категориальные характеристики в новые “dummy”/indicator features.

- Автоматически называет новые двоичные функции.

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

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

	size	price	classlabel	color_blue	color_green	color_red
0	S	10.1	shirt	0	1	0
1	M	13.5	pants	0	0	1
2	L	15.3	shirt	1	0	0

Кодирование со многими категориями (Encoding with many categories)

- Определите **иерархическую структуру**:

  Example: для почтового индекса можно попробовать использовать

  regions -> states -> city как иерархия,

  и можно выбрать конкретный уровень для кодирования функции почтового индекса.

- Group/bin категории в меньшее количество групп по сходству:

  Example: Для наборов демографических данных пользователей создайте возрастные группы: 1-15, 16-22, 23-30, и так далее.

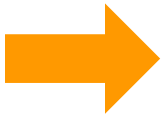
Кодирование со многими категориями (Encoding with many categories)

Target Encoding: Кодировать с использованием значений, которые могут объяснить цель.

Example: Усреднение целевого значения для каждой категории. Затем замените категориальные значения средним целевым значением.

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

x_1 -> cat a -> 3/5 = 0.6
x_1 -> cat b -> 0/2 = 0
x_2 -> cat c -> 1/2 = 0.5
x_2 -> cat d -> 2/5 = 0.4

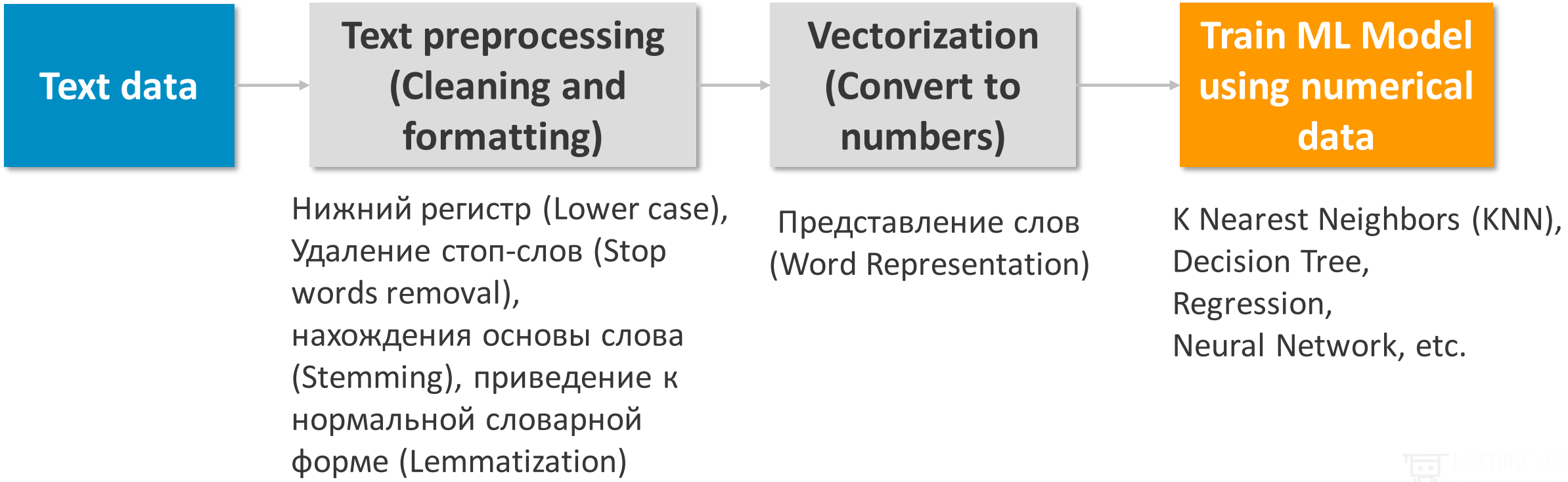


x_1	x_2	y
0.6	0.5	1
0.6	0.4	1
0	0.5	0
0.6	0.4	0
0.6	0.4	0
0.6	0.4	1
0	0.4	0

Text Preprocessing

Machine Learning with Text Data

- **Текст -** это распространенный тип данных, такой как заголовки, имена, обзоры или любой ввод произвольной формы.
- Модели машинного обучения нуждаются в **четко определенных числовых данных.**



Cleaning Text Data

- **Motivation:** в «грязном» тексте труднее найти закономерности.
  - Нормализация текста путем удаления шума: преобразование в нижний регистр, удаление пробелов, удаление специальных символов, удаление пометок и др.

  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”

Tokenization

- **Tokenization:** Разделяет текст на мелкие части по пробелам и знакам препинания.

Example:

Sentence	Tokens
“I don’t like eggs.”	“I”, “do”, “n’t”, “like”, “eggs”, “.”

Токены будут использоваться для дальнейшей очистки и векторизации.

Stop Words Removal

- **Stop Words**: Некоторые слова, которые часто встречаются в текстах, но не вносят особого вклада в общее значение.
- Common stop words: “a”, “the”, “so”, “is”, “it”, “at”, “in”, “this”, “there”, “that”, “my”, “by”, “nor”

Example:

Original sentence	Without stop words
“There is a tree near the house.”	“tree near house”

Stop Words Removal

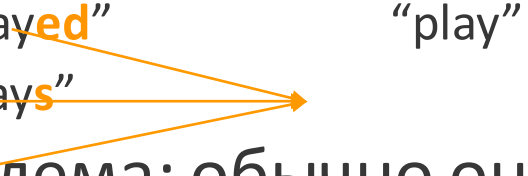
- **Stop Words** from the **Natural Language Tool Kit (NLTK)** library:

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', '
yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it'
s", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that',
"that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'a
t', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'bel
ow', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here
', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'suc
h', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "
don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "could
n't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", '
ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

Это хороший список стоп-слов для бинарной текстовой классификации отзывов о продуктах (положительный или отрицательный)?

Stemming

- Набор правил для разделения строки на подстроку, которая обычно имеет более общее значение.
  - Цель состоит в том, чтобы удалить аффиксы слов (особенно суффиксы), такие как “s”, “es”, “ing”, “ed”, etc.
    - “play**ing**”
    - “play**ed**”
    - ”play**s**”



  - Проблема: обычно он не работает с неправильными формами, такими как неправильные глаголы: **“taught”**, **“brought”**, etc.

Text Vectorization

Text Vectorization: Bag of Words

Модели машинного обучения нуждаются в **четко определенных числовых данных**.

‘Bag of Words’ (BoW) method:

- Преобразует текстовые данные в числовые функции.
- Называется **извлечением признаков**, поскольку мы извлекали важную информацию из исходного текста в числовой форме.
- Для каждого слова в документе мы получаем номер; может быть:
  - **binary** (1 or 0, присутствует или нет)
  - **количество слов или частота**

Bag of Words: Binary

Простой пример для **binary** функций:

	a	cat	dog	is	it	my	not	old	wolf
“It is a dog.”	1	0	1	1	1	0	0	0	0
“my cat is old”	0	1	0	1	0	1	0	1	0
“It is not a dog, it is a wolf.”	1	0	1	1	1	0	1	0	1

Bag of Words: Counts

Простой пример для **word counts** :

	a	cat	dog	is	it	my	not	old	wolf
“It is a dog.”	1	0	1	1	1	0	0	0	0
“my cat is old”	0	1	0	1	0	1	0	1	0
“It is not a dog, it is a wolf.”	2	0	1	2	2	0	1	0	1

Term Frequency (TF)

Term frequency (TF): Увеличивает веса общих слов в документе.

$$tf(term, doc) = \frac{number\ of\ times\ the\ term\ occurs\ in\ the\ doc}{total\ numbers\ of\ terms\ in\ the\ doc}$$

	a	cat	dog	is	it	my	not	old	wolf
“It is a dog.”	**0.25**	0	**0.25**	**0.25**	**0.25**	0	0	0	0
“my cat is old”	0	**0.25**	0	**0.25**	0	**0.25**	0	**0.25**	0
“It is not a dog, it is a wolf.”	**0.22**	0	**0.11**	**0.22**	**0.22**	0	**0.11**	0	**0.11**

Inverse Document Frequency (IDF)

Inverse document frequency (IDF):
Уменьшает вес для часто используемых слов и увеличивает вес для редких слов в словаре.

$$idf(term) = log\left(\frac{n_{documents}}{n_{documents\ containing\ the\ term} + 1}\right) + 1$$

Example:

$$idf(\text{"}cat\text{"}) = 1.18$$
$$idf(\text{"}is\text{"}) = 0.87$$

term	idf
a	$log(3/3) + 1 = 1$
cat	$log(3/2) + 1 = 1.18$
dog	$log(3/3) + 1 = 1$
is	$log(3/4) + 1 = 0.87$
it	$log(3/3) + 1 = 1$
my	$log(3/2) + 1 = 1.18$
not	$log(3/2) + 1 = 1.18$
old	$log(3/2) + 1 = 1.18$
wolf	$log(3/2) + 1 = 1.18$

Inverse Doc. Freq. (TF-IDF)

Term Freq. Inverse Doc. Freq (TF-IDF): Сочетает в себе частоту термина и обратную частоту документа.

$$tf_{idf}(term, doc) = tf(term, doc) * idf(term)$$

	a	cat	dog	is	it	my	not	old	wolf
“It is a dog.”	**0.25**	0	**0.25**	**0.22**	**0.25**	0	0	0	0
“my cat is old”	0	**0.3**	0	**0.22**	0	**0.3**	0	**0.3**	0
“It is not a dog, it is a wolf.”	**0.22**	0	**0.11**	**0.19**	**0.22**	0	**0.13**	0	**0.13**

Bag of Words in sklearn

CountVectorizer: **sklearn** text vectorizer, преобразует набор текстовых документов в матрицу количества токенов - .fit(), .transform()

```
from sklearn.feature_extraction.text import CountVectorizer
countVectorizer = CountVectorizer(binary=True)

sentences = ['This is the first document.',
             'This is the second document.',
             'and the third one.',
            ]

X = countVectorizer.fit_transform(sentences)
print(X.toarray())
```

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

```
[[0 1 1 1 0 0 1 0 1]
 [0 1 0 1 0 1 1 0 1]
 [1 0 0 0 1 0 1 1 0]]
```

Bag of Words in sklearn

TfidfVectorizer: **sklearn** text vectorizer, преобразует набор текстовых документов в матрицу TF-IDF признаков - .fit(), .transform()

- Возвращает нормализованную матрицу частот терминов, когда“use_idf = False”:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(use_idf=False)
```

- Возвращает более гладкую матрицу TF-IDF, когда“use_idf = False”:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(use_idf=True)
```

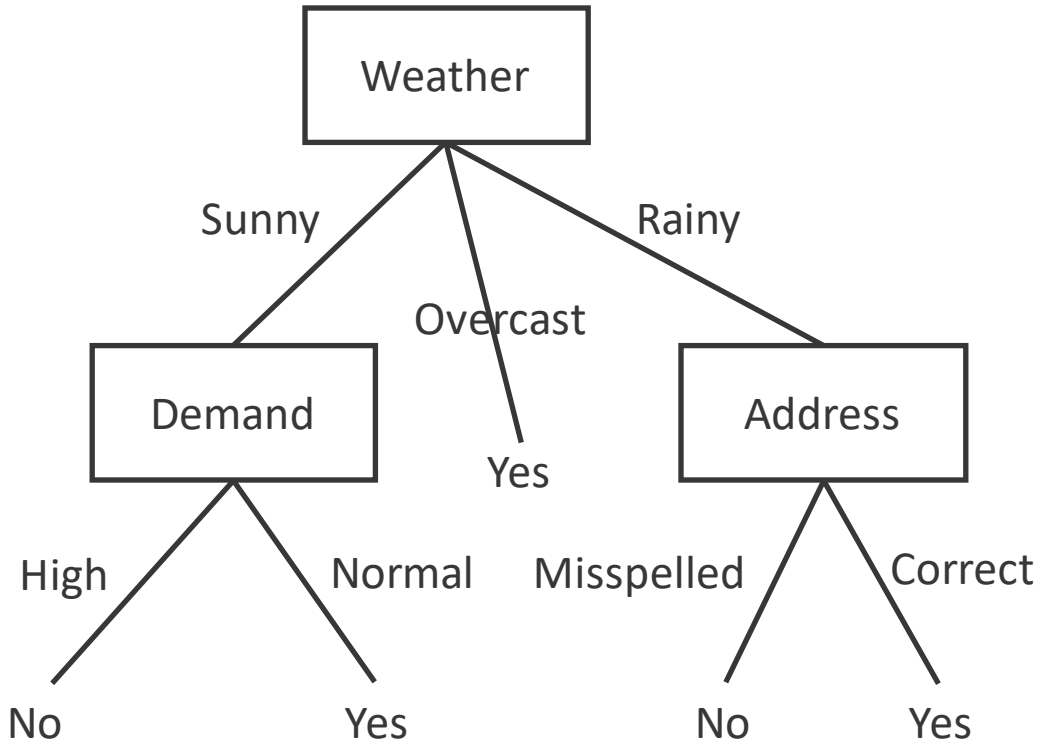
Tree-based Models

Problem: Прогноз доставки посылки

- Используя этот датасет, давайте спрогнозируем своевременную доставку посылки (да/нет), используя **Decision Tree**.
- **Итеративно разделите** набор данных на подмножества (ветви) так, чтобы конечные подмножества (листья) содержали в основном один класс.

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

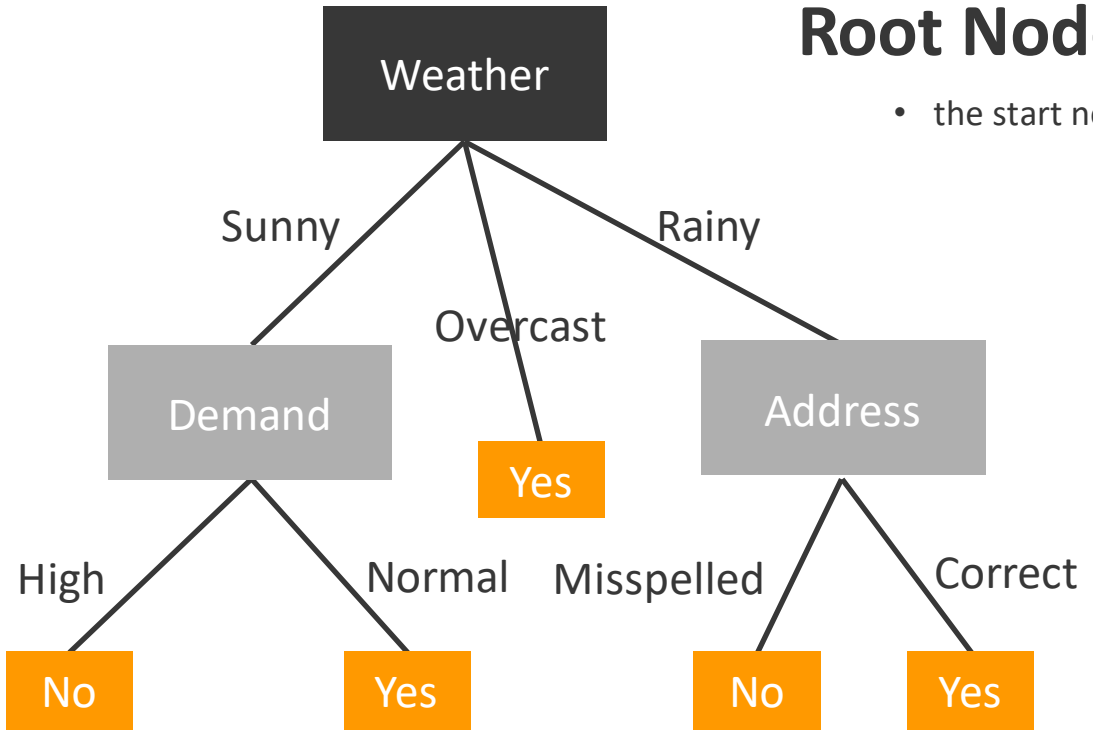
ML Model: Decision Tree



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

Decision Деревья представляют собой структуры, похожие на блок-схемы, которые можно использовать для задач классификации или регрессии.



Root Node

- the start node

Как обучить дерево решений?

Internal Nodes

- ровно один входящий узел и два или более исходящих ребра
- имеет атрибутные условия для разделения записей

Leaf or Terminal Nodes

- ровно одно входящее ребро и никаких исходящих ребер
- ему присваивается метка класса (классификация) или значение (регрессия)

Learn a Decision Tree

Алгоритм ID3*:

(Повторите шаги ниже)

1. **Выбрать** «лучшую функцию» для разделения (мы увидим, как выбрать)
2. **Разделить** обучающие выборки в соответствии с выбранной функцией.
3. **Остановиться,** если у нас есть примеры одного класса или если мы использовали все функции, и отметьте его как листовой узел.

Подход «сверху вниз»: рост дерева от корневого узла к листовым узлам.

*ID3 (Iterative Dichotomiser 3)

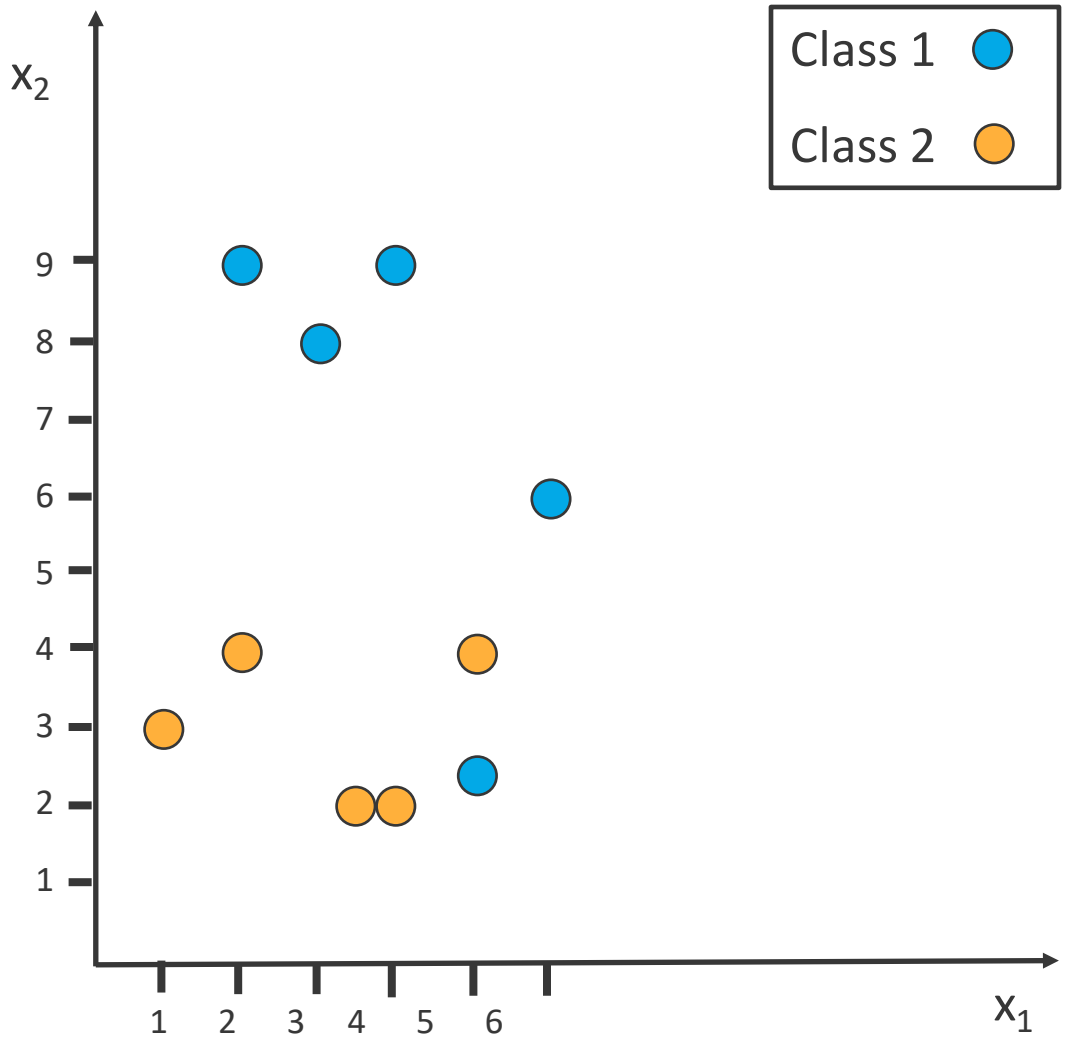
Decision Trees: Numerical Example

- По набору данных, давайте спрогнозируем класс y (1 или 2), используя **Decision Tree**.

- **Итеративно разделите** набор данных на подмножества от корневого узла, чтобы конечные узлы содержали в основном один класс (как можно более чистый).

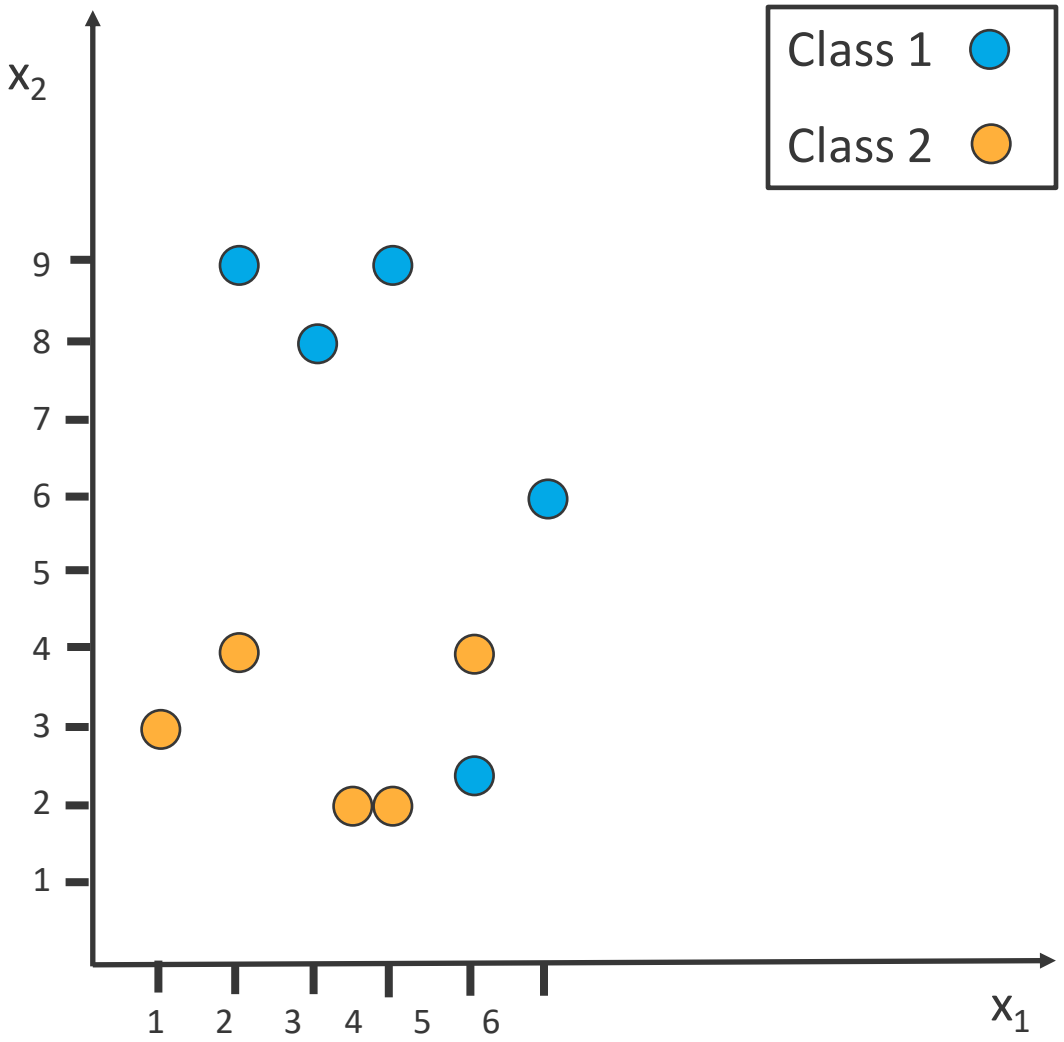
x_1	x_2	y
3.5	2	1
5	2.5	2
1	3	1
2	4	1
4	2	1
6	6	2
2	9	2
4	9	2
5	4	1
3	8	2

Decision Trees: Numerical Example



x_1	x_2	y
3.5	2	1
5	2.5	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

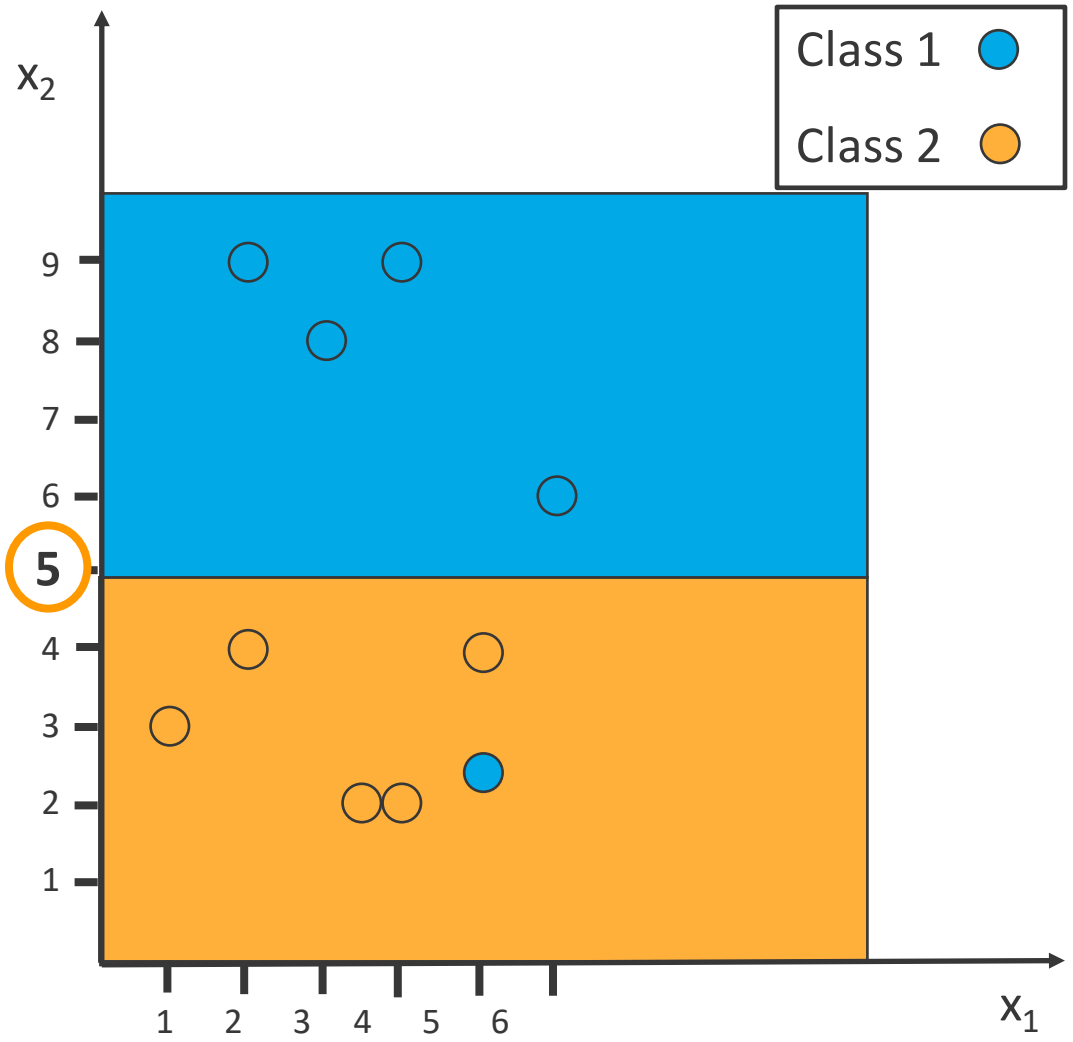


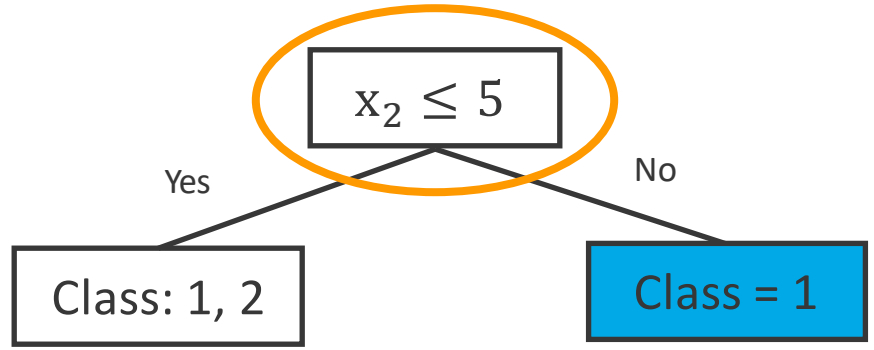
Class: 1, 2

Какую функцию (x1 или x2) использовать для разделения этого набора данных, чтобы лучше всего отделить класс 1 от класса 2?

[выберите разбиения так, чтобы потомки были «чище», чем их родители]

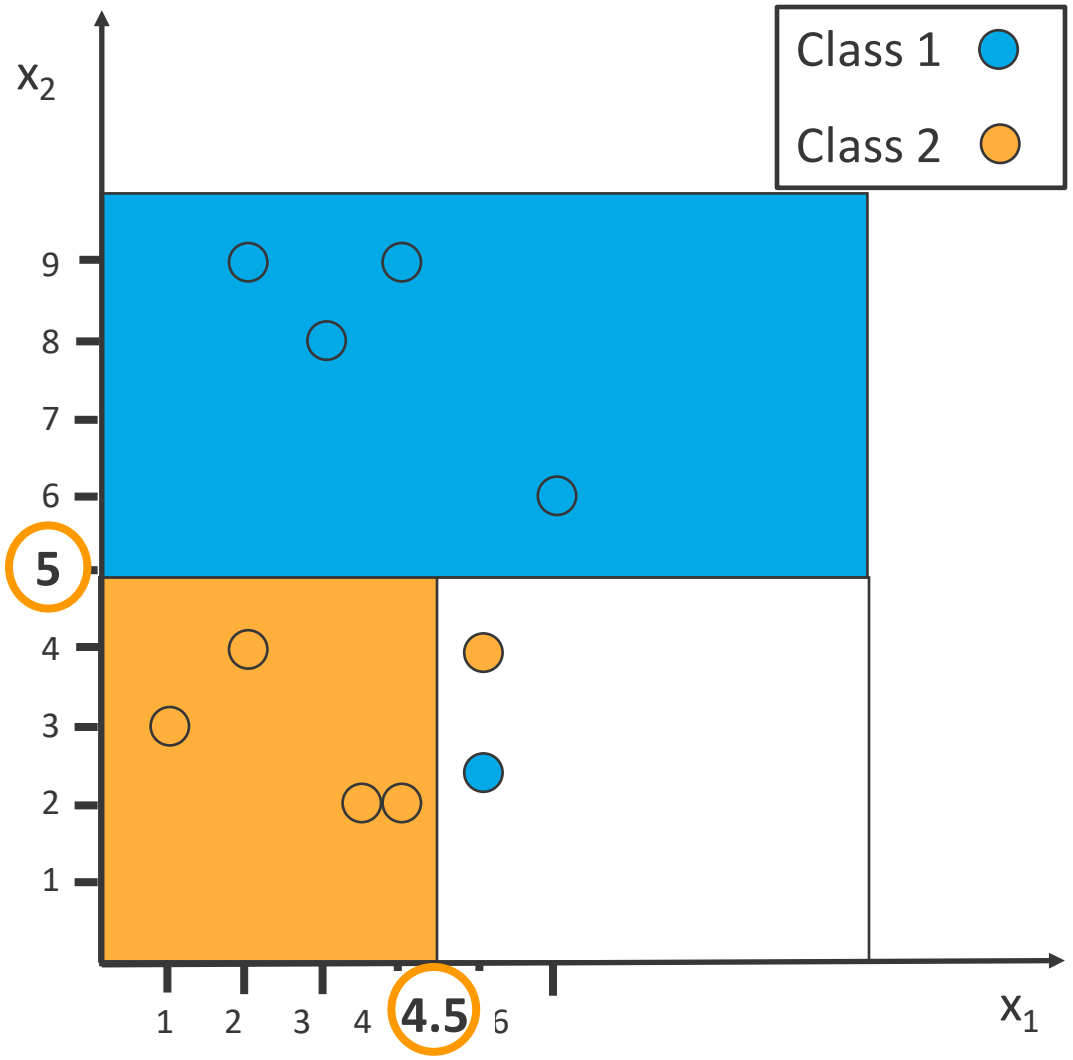
Decision Trees: Numerical Example

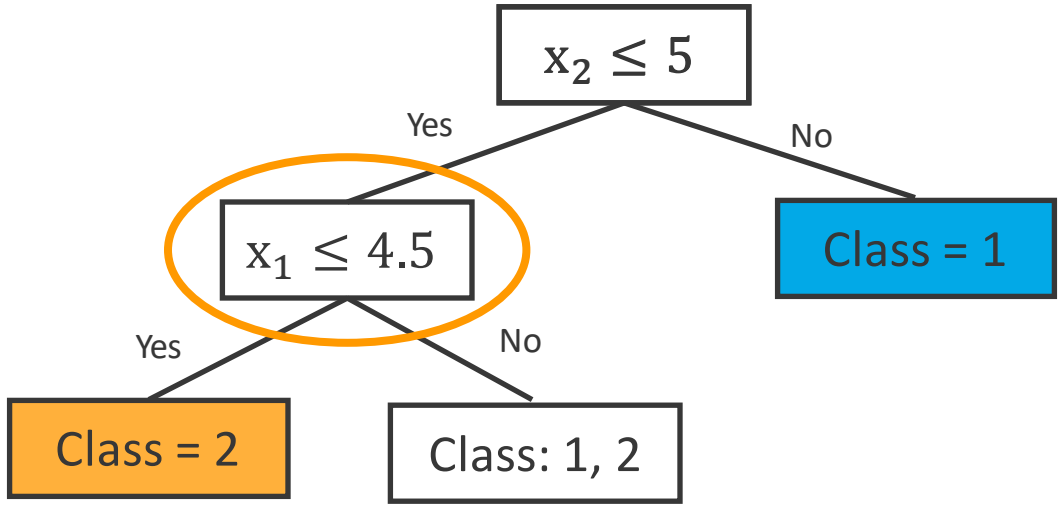




Какую функцию (x1 или x2) использовать для разделения этого подмножества, чтобы лучше всего отделить класс 1 от класса 2?

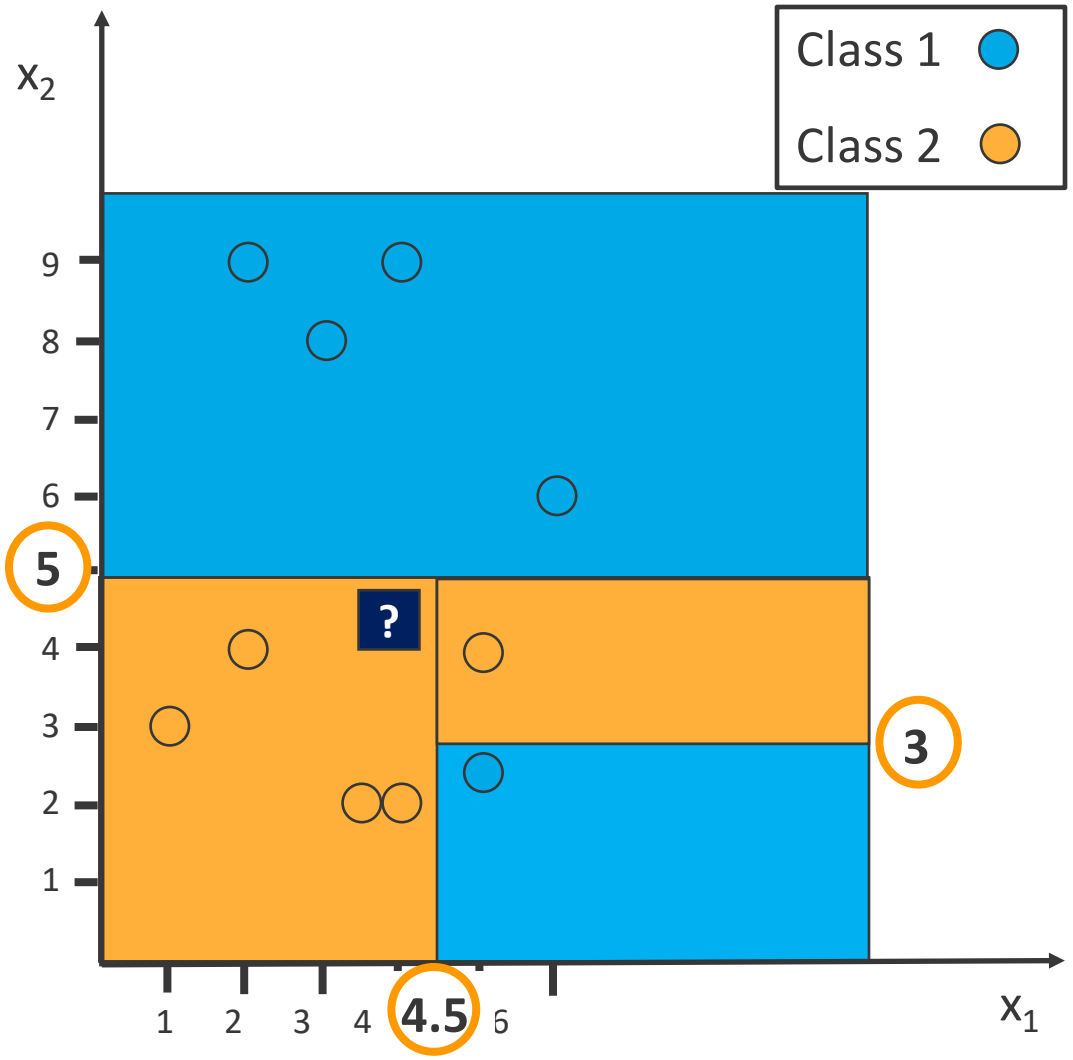
Decision Trees: Numerical Example

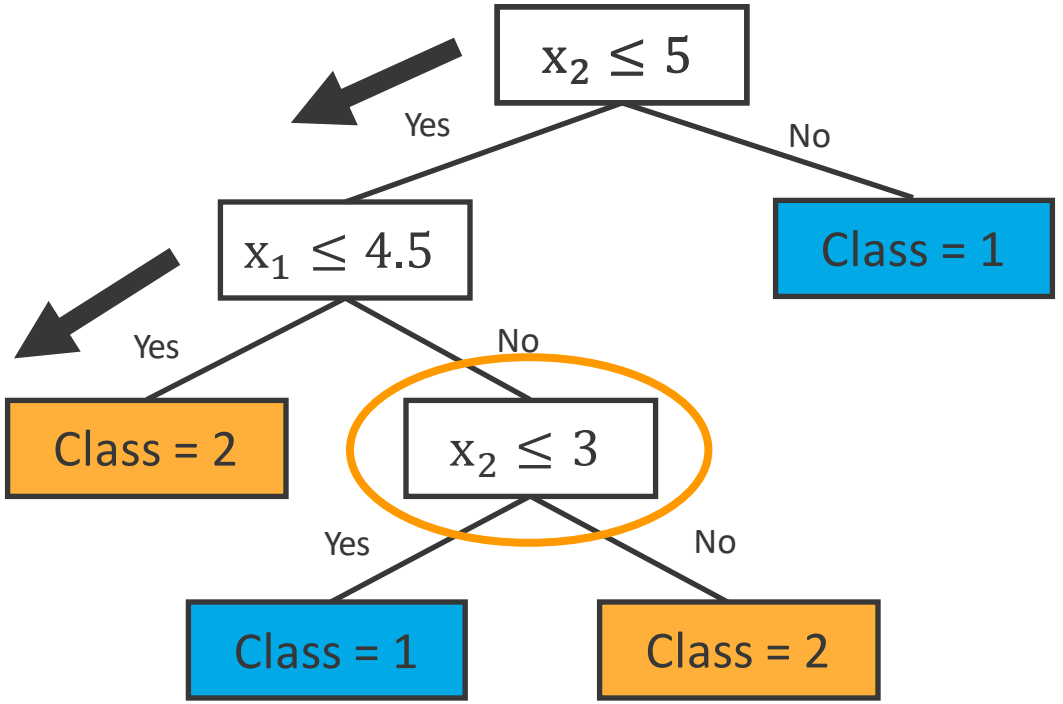




Какую функцию (x1 или x2) использовать для разделения этого подмножества, чтобы лучше всего отделить класс 1 от класса 2?

Decision Trees: Numerical Example





Decision Trees: Example

[9+, 5-]

Class: Yes, No

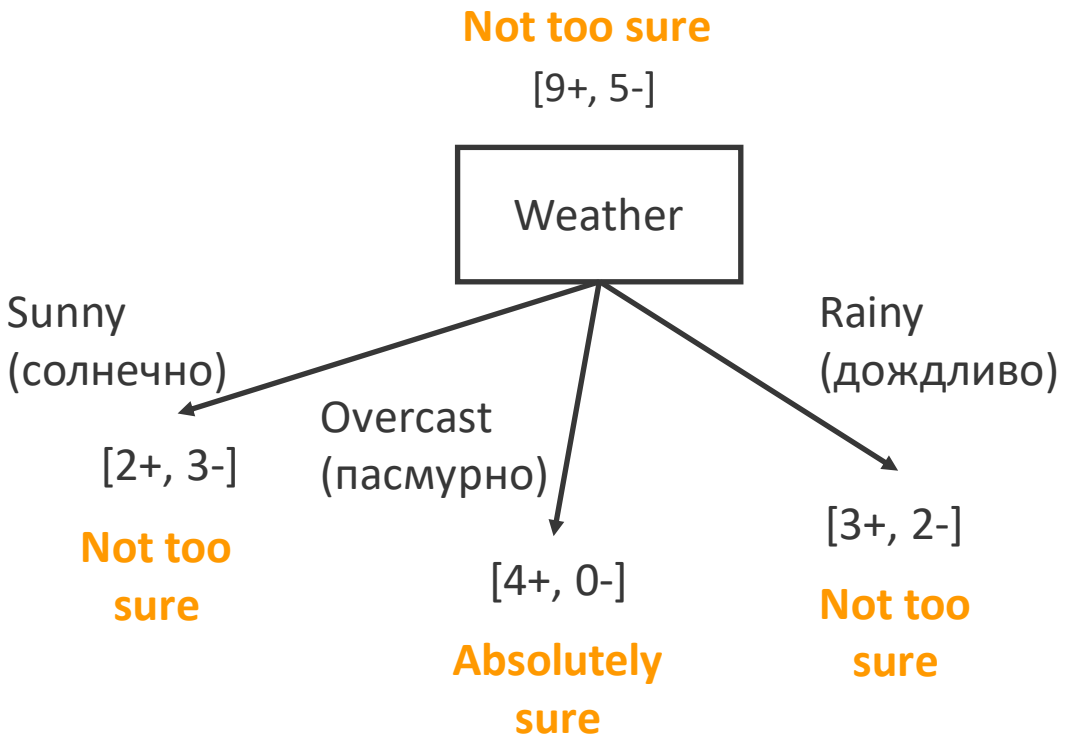
Какую функцию («Погода», «Спрос» или «Адрес») использовать для разделения набора данных, чтобы лучше всего отделить класс «Нет» от класса «Да»?

[выберите разбиения так, чтобы потомки были «чище», чем их родители]

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

Best Feature to Split with?

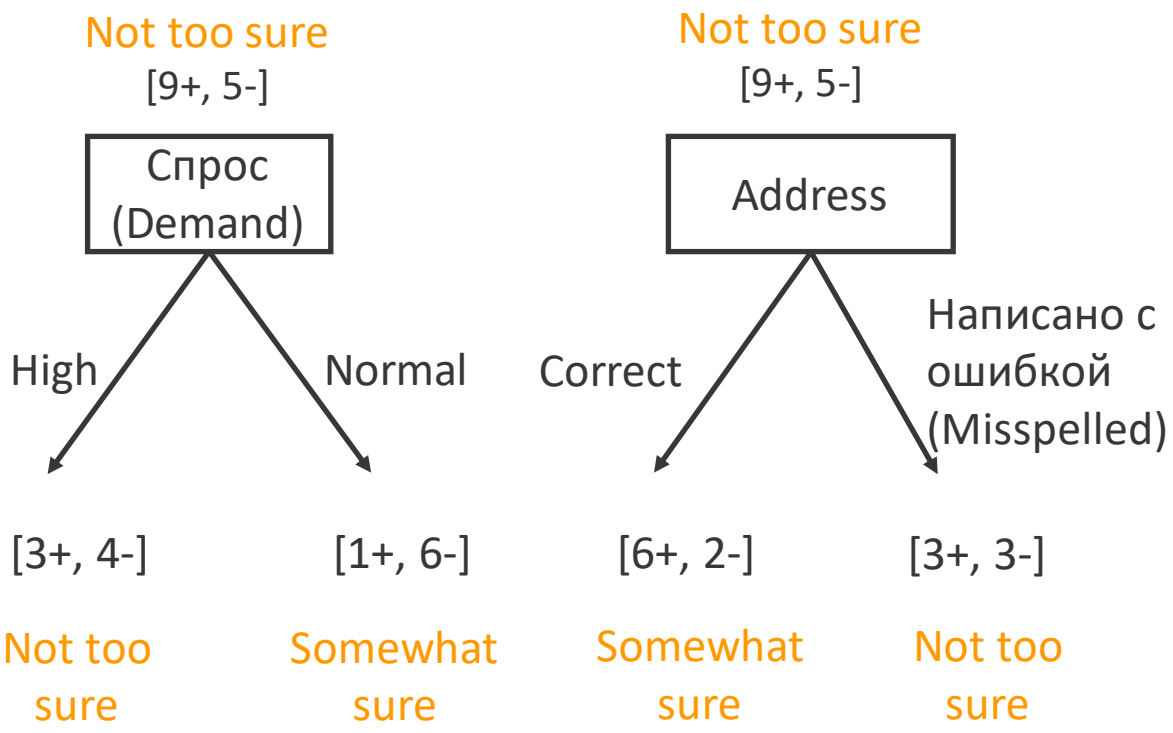
Хорошее разделение приводит к общей меньшей неопределенности (энтропии, примеси). Например:



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

Какая лучшая функция (признак) для разделения?

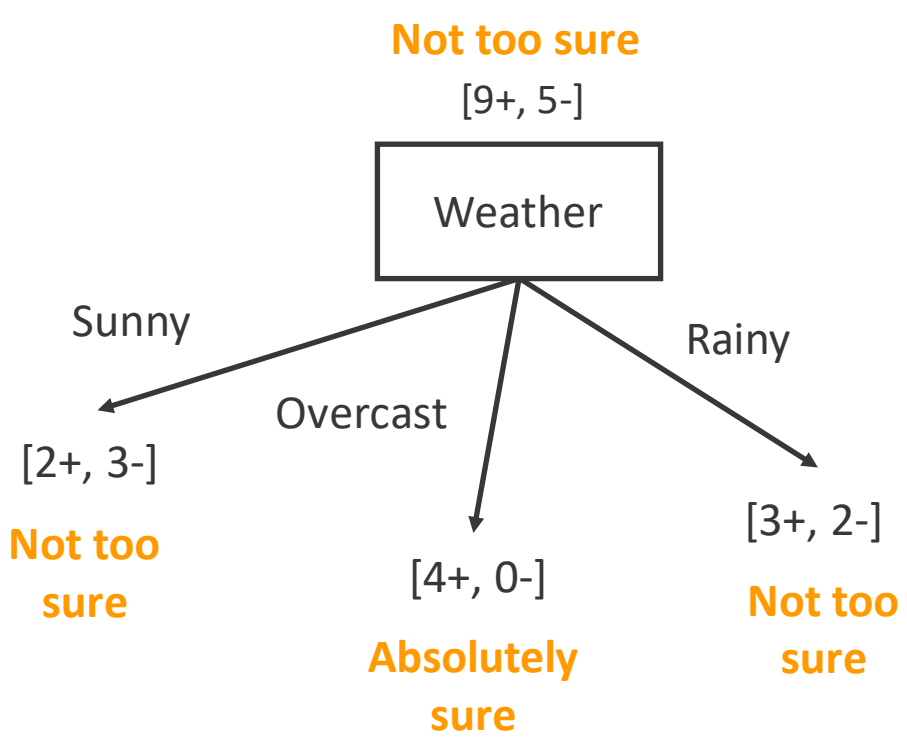
Хорошее разделение приводит к общей меньшей неопределенности (энтропии, примеси). Например:

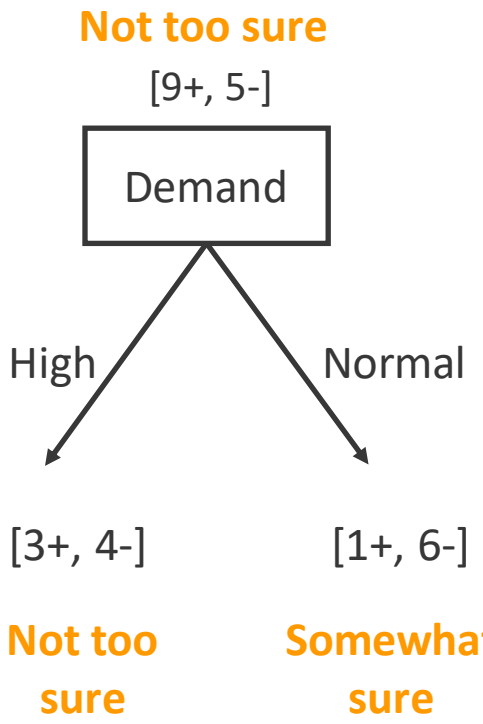


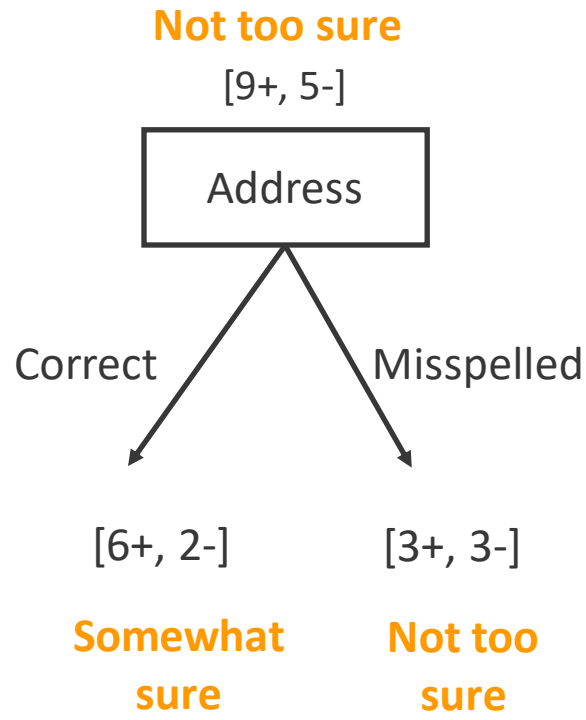
Weather	Demand	Address	ontime
Sunny	High	Correct	No
Sunny	High	Misspelled	No
Overcast	High	Correct	Yes
Rainy	High	Correct	Yes
Rainy	Normal	Correct	Yes
Rainy	Normal	Misspelled	No
Overcast	Normal	Correct	Yes
Sunny	High	Correct	No
Sunny	Normal	Correct	Yes
Rainy	Normal	Misspelled	Yes
Sunny	Normal	Misspelled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

Best Feature to Split with?

Какое разделение приведет к уменьшению общей неопределенности (примеси)?

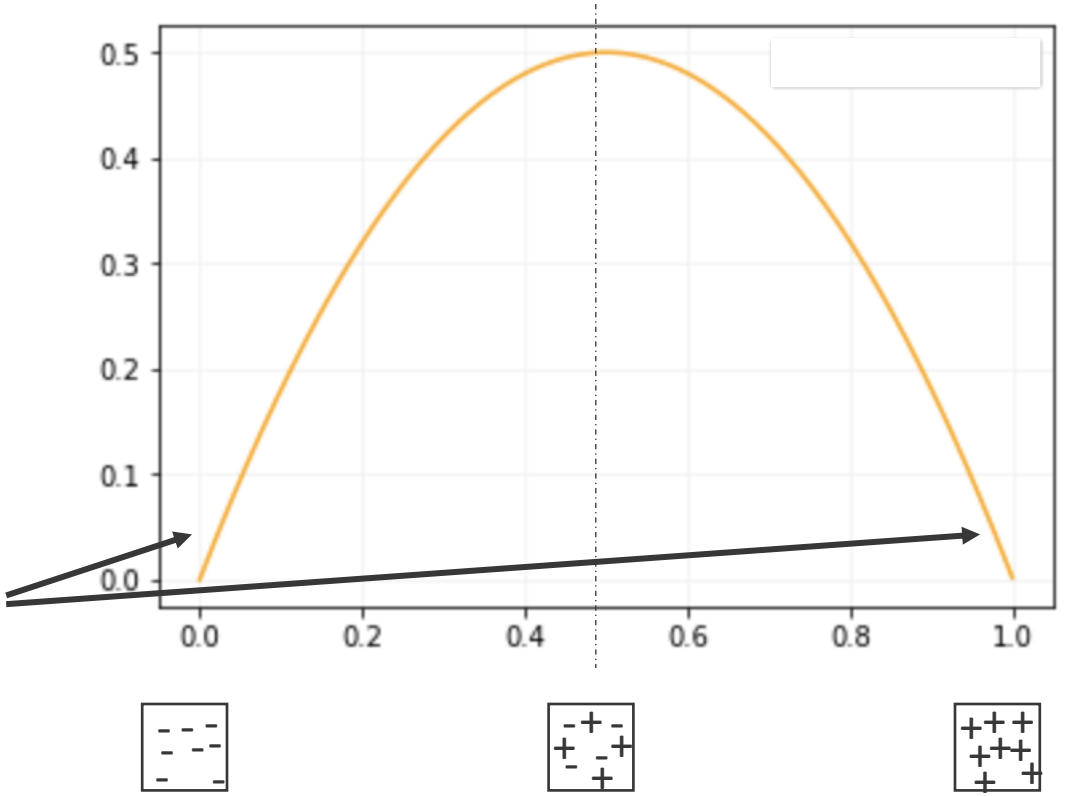






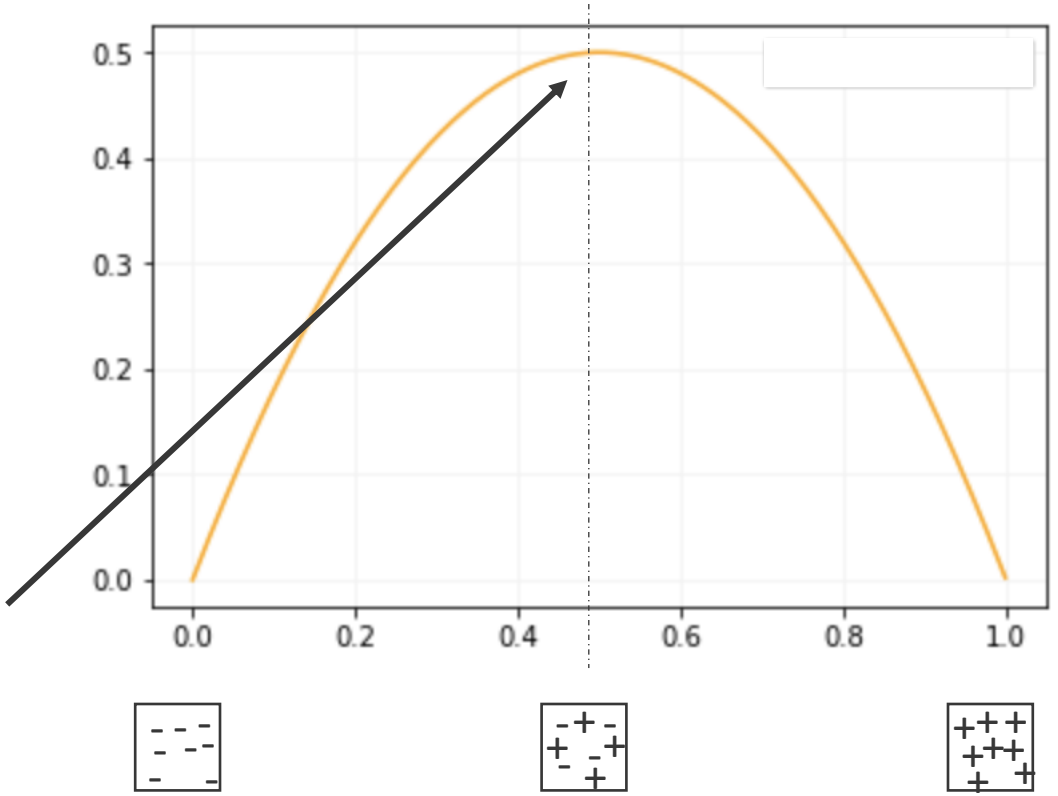
Как измерить неопределенность

Если у нас только + образцы
или только - образцы:
Низкая неопределенность



Как измерить неопределенность

Если у нас есть смесь + и - образцов:
Высокая неопределенность



Как измерить неопределенность

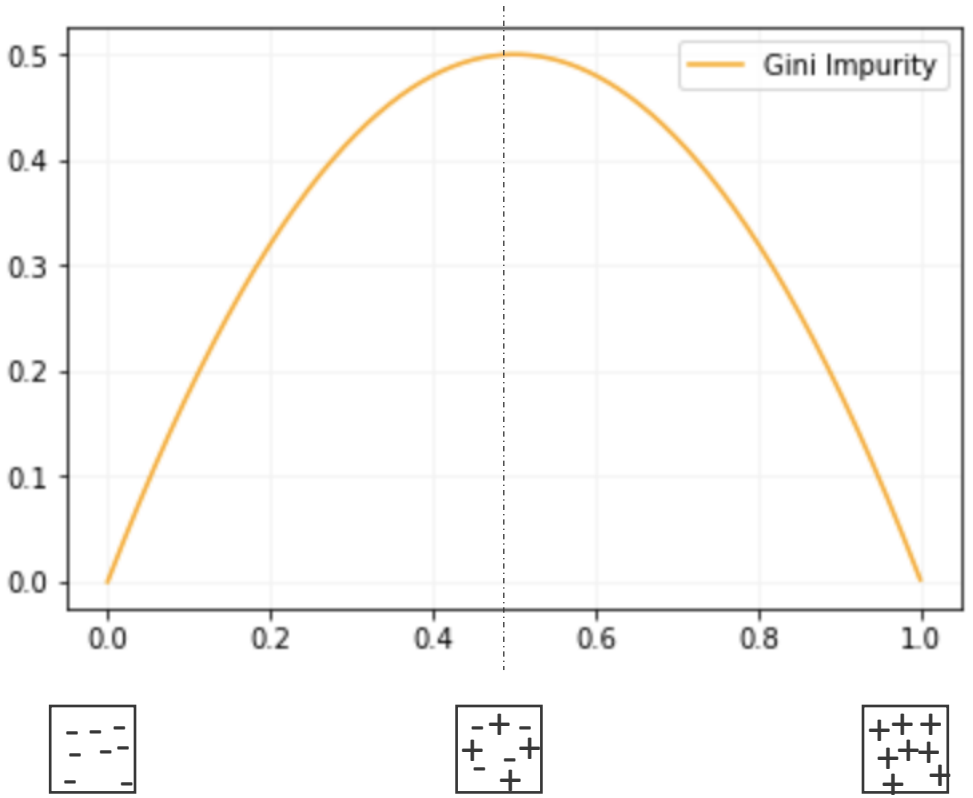
Мы будем использовать примесь (неопределенность, критерий) Джини:

$$i(p_1, ..., p_k) = \sum_{k=1}^{n} p_k(1 - p_k)$$

(n: число классов, p_k: доля объектов класса k (вероятность) выбора примера из класса)

Еще одна мера:

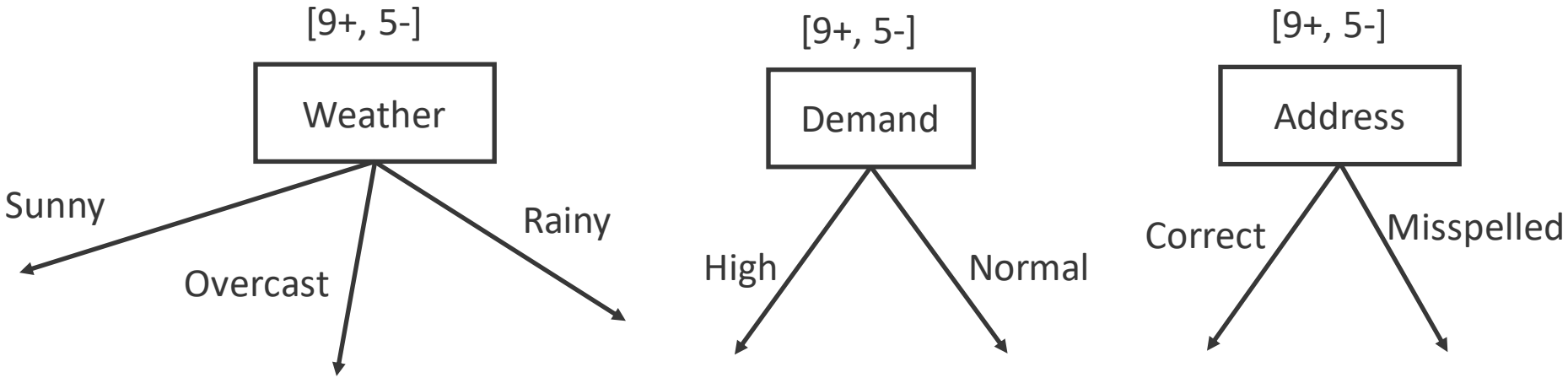
Энтропия (Entropy): More details



Прирост информации (Information Gain) & Feature Selection

Прирост информации (Information Gain): Ожидаемое снижение неопределенности благодаря выбранной функции.

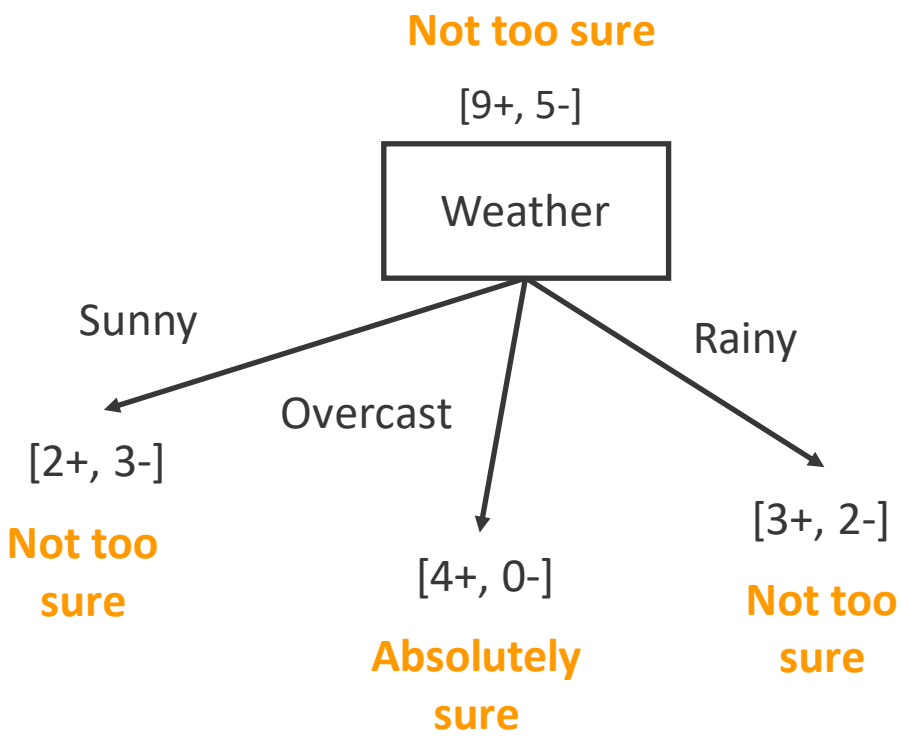
$$\text{Gain} = \underbrace{i(p_1, \ldots, p_k)_{before}}_{\text{Impurity before split}} - \underbrace{i(p_1, \ldots, p_k)_{after}}_{\text{Impurity after split}}$$



«Погода», «Спрос» или «Адрес», что мы должны выбрать в качестве функции для разделения?

Вычисление примеси Джини (Calculating Gini Impurity)

$$i_{before} = 0.46$$

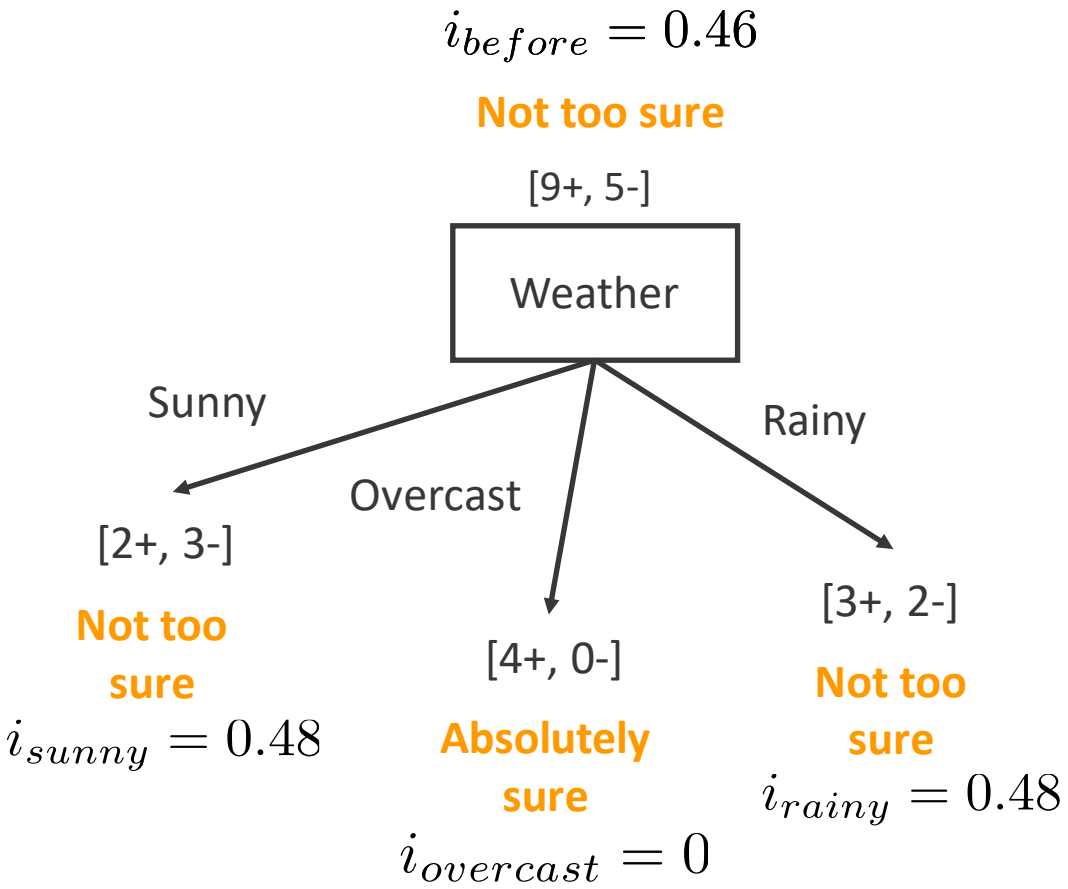


Примесь (impurity) **Gini**:

$$i(p_1, ..., p_k) = \sum_{k=1}^{n} p_k(1 - p_k)$$

$$i_{before} = \frac{9}{14} * \frac{5}{14} + \frac{5}{14} * \frac{9}{14} = 0.46$$

Вычисление примеси Джини (Calculating Gini Impurity)



Примесь (impurity) **Gini**:

$$i(p_1, ..., p_k) = \sum_{k=1}^{n} p_k(1 - p_k)$$

$$i_{sunny} = \frac{2}{5} * \frac{3}{5} + \frac{3}{5} * \frac{2}{5} = 0.48$$

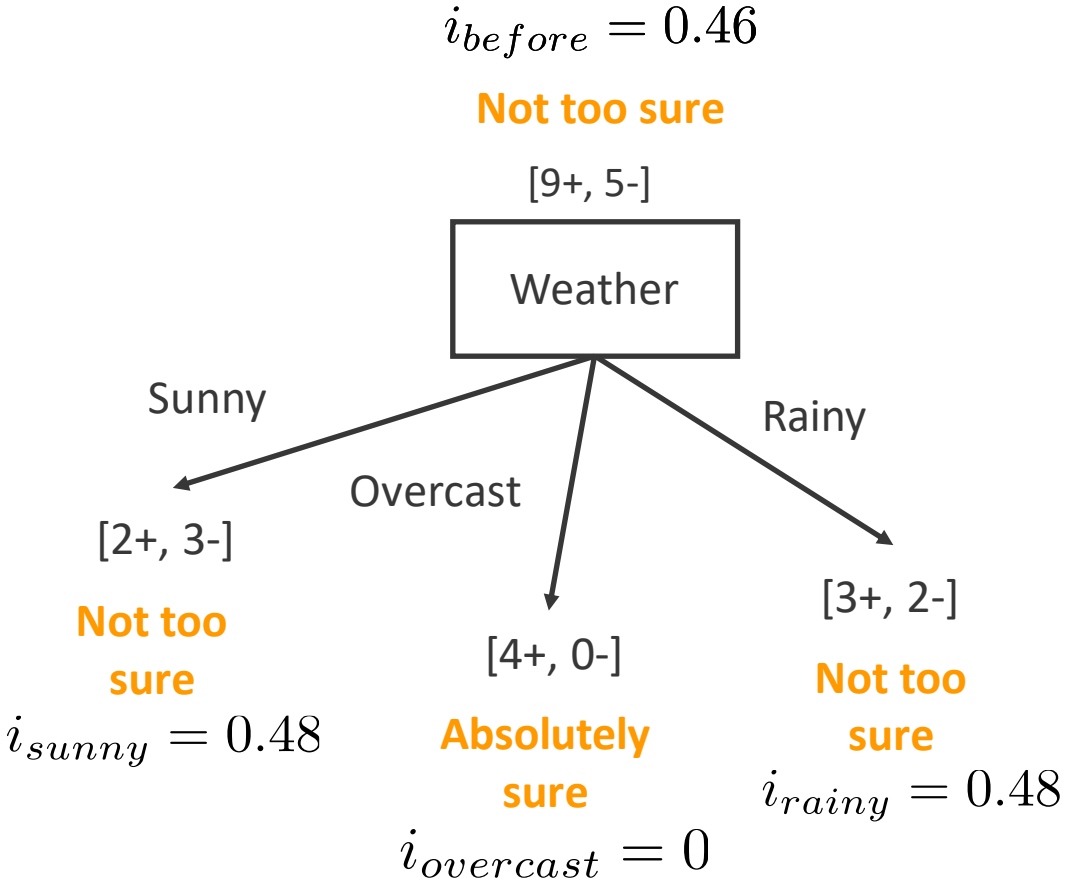
$$i_{overcast} = \frac{4}{4} * \frac{0}{4} + \frac{0}{4} * \frac{4}{4} = 0$$

$$i_{rainy} = \frac{3}{5} * \frac{2}{5} + \frac{2}{5} * \frac{3}{5} = 0.48$$

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

$$= 0.34$$ Взвешенная сумма примеси (Weighted sum of impurities)

Прирост информации (Information Gain) & Feature Selection

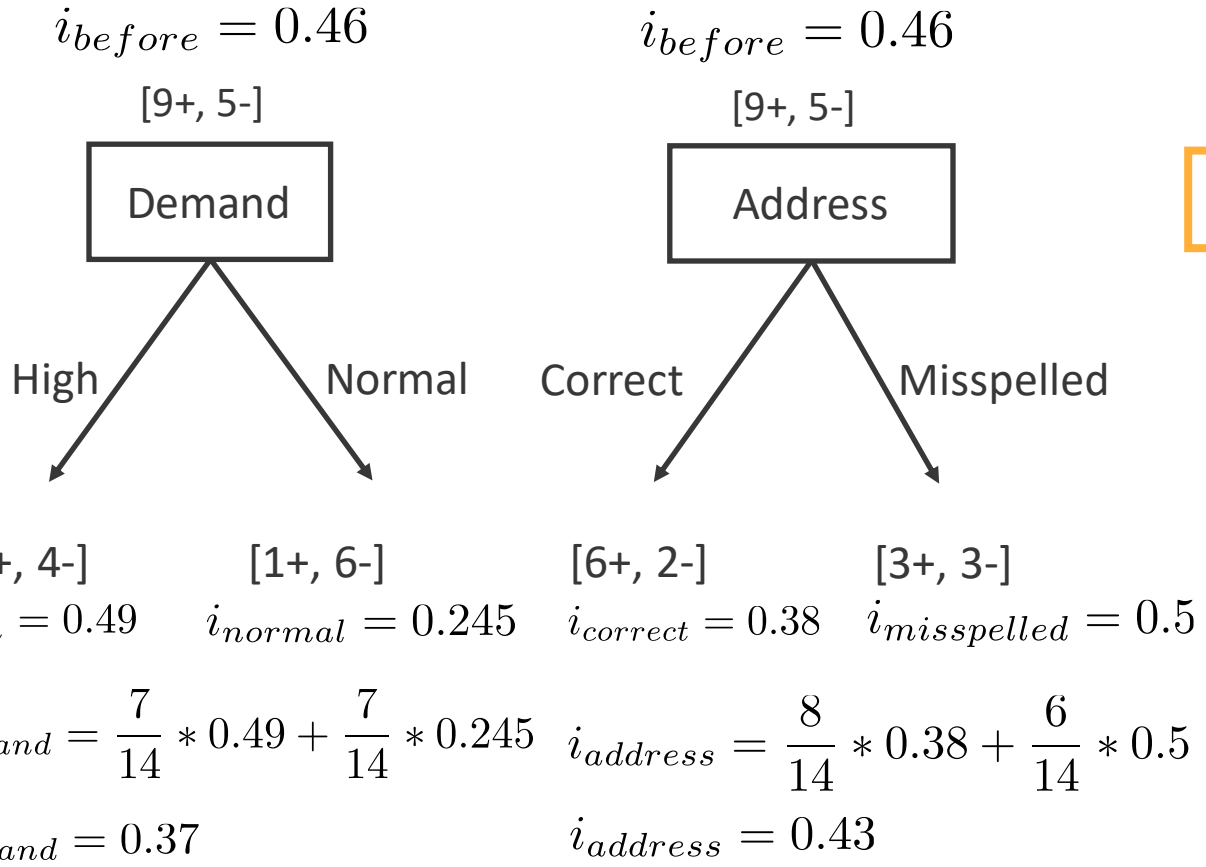


Gain = $\underbrace{i(p_1, \ldots, p_k)_{before}}_{\text{Impurity before split}} - \underbrace{i(p_1, \ldots, p_k)_{after}}_{\text{Impurity after split}}$

$i_{before} = 0.46$ $\quad$ $i_{weather} = 0.34$

Прирост (Gain) “Weather” = 0.46-0.34=0.12

Information Gain & Feature Selection

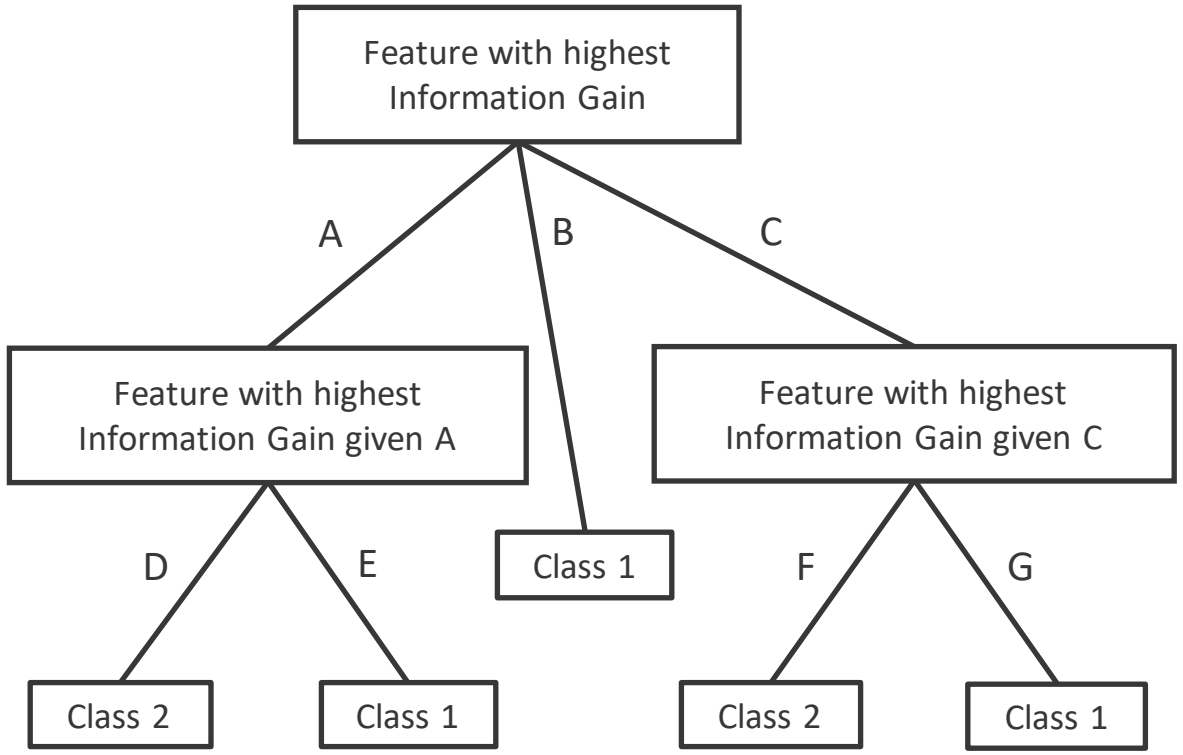


Comparing gains for each feature:

Gain(“Weather”) = 0.46-0.34 = 0.12
Gain(“Demand”) = 0.46-0.37 = 0.09
Gain(“Address”) = 0.46-0.43 = 0.03

«Погода» имеет самый высокий прирост из всех, поэтому мы начинаем дерево с функцией «Погода» в качестве корневого узла!

Recap ID3 Algorithm



Алгоритм ID3*:

1. **Выбрать** «лучшую функцию» для разделения.
2. **Разделить** обучающие выборки в соответствии с выбранной функцией.
3. **Остановиться,** если у нас есть примеры одного класса или если мы использовали все функции, и отметьте его как листовой узел.
4. **Присвоить** листу класс большинства примеров в нем.

*Чтобы построить регрессор дерева решений: 1. Замените прирост информации уменьшением стандартного отклонения. 3. Остановитесь, когда числовые значения однородны (стандартное отклонение равно нулю) или если использовались все функции, и отметьте его как листовой узел. 4. Назначьте листу-узлу среднее значение его выборок.

Decision Trees in sklearn

DecisionTreeClassifier: **sklearn** Decision Tree classifier (есть также версия Regressor) - .fit(), .predict()

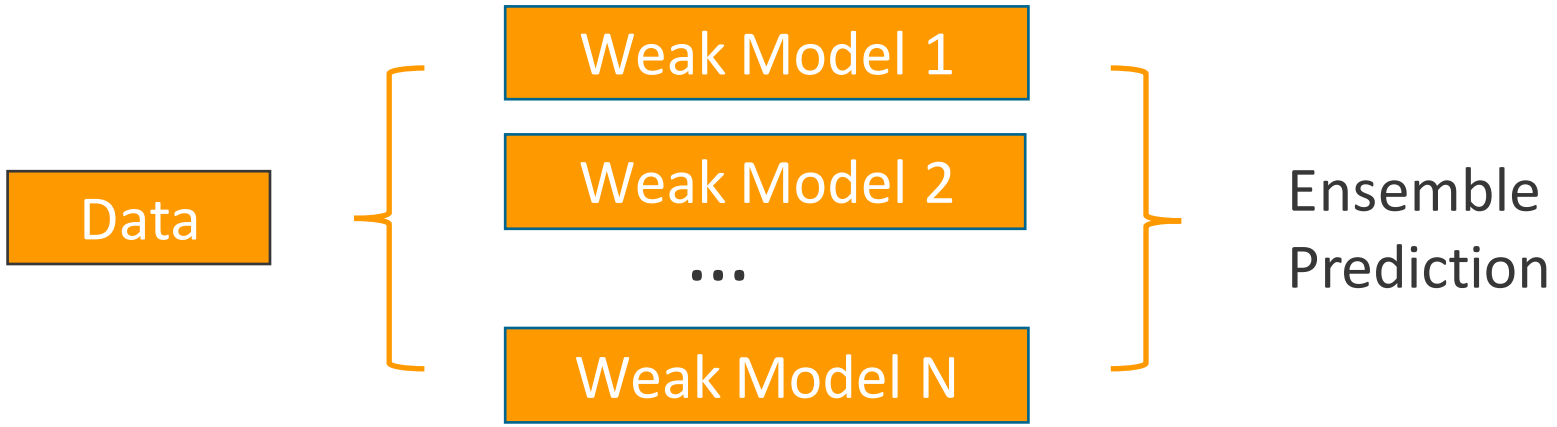
DecisionTreeClassifier(*criterion='gini'*,
max_depth=None, *min_samples_split=2*,
min_samples_leaf=1, *class_weight=None*)

Полный список параметров гораздо больше.

Ensemble Methods: Bagging

Ансамблевое обучение (Ensemble Learning)

Ансамблевые методы создают сильную модель, комбинируя прогнозы **нескольких слабых моделей** (также называемых **слабыми обучающимися или базовыми оценками**), построенных с заданным набором данных и заданным алгоритмом обучения.



Будем рассматривать модели ансамбля Bagging и Boosting.

Bagging (Bootstrap Aggregating)

Bagging (Bootstrap Aggregating) method:

- Произвольно возьмите N образцов фиксированного размера из обучающего набора (с заменой) - метод **bootstrap**

Example: given data [1, 2, 3, 4, 5, 6, 7, 8, 9], образцы размера 6:

[1, 1, 2, 4, 9, 9]; [2, 4, 5, 5, 7, 7]; [1, 1, 1, 1, 1, 1]; [1, 2, 4, 5, 7,9]

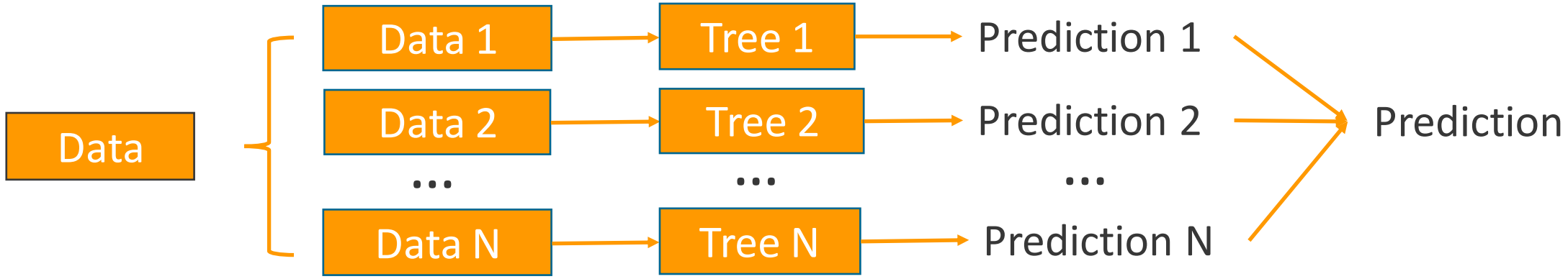
- Создавайте **независимые** оценщики (модели) одного типа для каждого подмножества
- Результат работы ансамбля: голосование по большинству (или взвешенное) или усреднение прогнозов всех оценщиков (для задачи регрессии).

Реализация концепции Bagging Decision Trees: **Random Forest**

Bagging trees: Random Forest

Random Forest: Bagging Decision Trees

- Создайте случайные подмножества (с заменой) из исходного набора данных
- Постройте **дерево решений** для каждого начального подмножества
- Объедините прогнозы из каждого дерева для окончательного прогноза



Random Forest in sklearn

RandomForestClassifier: **sklearn** Random Forest classifier (есть версия для Regressor) - .fit(), .predict()

RandomForestClassifier(*n_estimators=100,*
max_samples=None, max_features='auto',
criterion='gini', max_depth=None, min_samples_split=2,
min_samples_leaf=1, class_weight=None)

Полный список параметров гораздо больше.

Bagging in sklearn

BaggingClassifier: sklearn общий интерфейс для bagging который может быть предоставлен любым ***base_estimator*** - .fit(), .predict()

BaggingClassifier(*base_estimator=None, n_estimators=10, max_samples=1.0, bootstrap=True*)

Полный список параметров гораздо больше.

Hyperparameter Tuning

Настройка гиперпараметров (Hyperparameter Tuning)

- **Hyperparameters** это **параметры алгоритмов машинного о**бучения, которые влияют на структуру алгоритмов и производительность (качество) моделей.

  Примеры hyperparameters:

  - **K Nearest Neighbors**: n_neighbors, metric
  - **Decision trees:** max_depth, min_samples_leaf, class_weight, criterion
  - **Random Forest**: n_estimators, max_samples
  - **Ensemble Bagging:** base_estimator, n_estimators

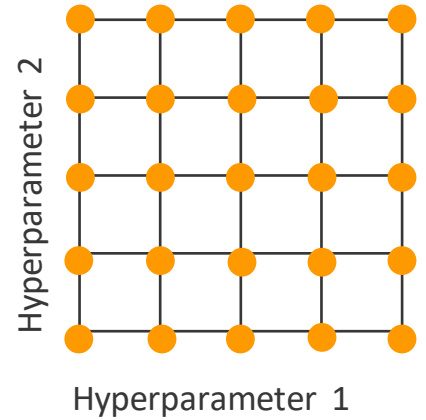
- **Hyperparameter tuning** (настройка гиперпараметров) ищет лучшую комбинацию гиперпараметров (комбинацию, которая максимизирует качество модели).

Grid Search in sklearn

GridSearchCV: **sklearn** базовый метод hyperparameter tuning, находит оптимальное сочетание гиперпараметров путем перебора в течение определенных значений параметров -

.fit(), .predict()

GridSearchCV(*estimator*, *param_grid*, *scoring=None)*



Example: Hyperparameters for a Decision Tree:

param_grid ={max_depth: [5, 10, 50, 100, 250],
min_samples_leaf: [15, 20, 25, 30, 35]}

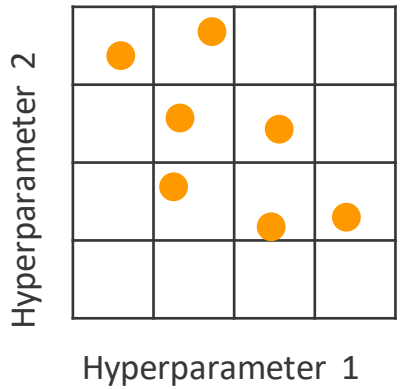
Total hyperparameters combinations
5 x 5 = 25

[5, 15], [5, 20], [5, 25], [10, 15], …

Randomized Search in sklearn

RandomizedSearchCV: рандомизированный поиск по гиперпараметрам

- Выбирает фиксированное количество (заданное параметром n_iter) случайных комбинаций значений гиперпараметров и пробует только их.
- Возможна выборка из распределений (используется выборка с заменой), если в качестве распределения задан хотя бы один параметр.



RandomizedSearchCV(*estimator*, *param_distributions*,

n_iter=10, scoring=None)

Example: Hyperparameters for a Decision Tree:

param_grid ={max_depth: [5, 10, 50, 100, 250],
min_samples_leaf: uniform(15,35,5)}

Байесовский поиск (Bayesian Search)

- Метод байесовского поиска (**Bayesian Search**) отслеживает предыдущие оценки гиперпараметров и строит вероятностную модель.
- Он пытается сбалансировать **исследование** (неопределенный набор гиперпараметров) и **реализацию** (гиперпараметры с хорошей вероятностью должны быть оптимальными).
- Предпочитает точки рядом с теми, которые хорошо сработали.

Data Preprocessing with Pipeline (sklean)

Transformers in sklearn

- **SimpleImputer, StandardScaler, MinMaxScaler, LabelEncoder, OrdinalEncoder, OneHotEncoder**, и **CountVectorizer** принадлежат к классу **transformers** sklearn, у всех есть:
  - **.fit()** method: изучает преобразование из **обучающего** набора данных
  - **.transform()** method: применяет преобразование к любому набору данных (**training, validation, test**) для предварительной обработки

К обучающему набору также можно применить **.fit_transform()**

ColumnTransformer in sklearn

ColumnTransformer: применяет преобразователи к столбцам массива или pandas DataFrame – `.fit(), .transform()`

- Позволяет **отдельно преобразовывать** разные столбцы или подмножества входных столбцов (числовые, категориальные, текстовые).
- Функции, генерируемые каждым преобразователем, будут **объединены**, чтобы сформировать **единое пространство функций**.
- Это полезно для смешанных наборов табличных данных, чтобы объединить несколько механизмов извлечения признаков или преобразований в один преобразователь.

ColumnTransformer and Pipeline

```
numerical_processing = Pipeline([
   (‘num_imputer’, SimpleImputer(strategy='mean’)),
   (‘num_scaler’, MinMaxScaler())])

categorical_processing = Pipeline([
   (‘cat_imputer’, Imputer(strategy='constant', fill_value='missing’)),
   (‘cat_encoder’, OneHotEncoder(handle_unknown='ignore’))])

processor = ColumnTransformer(transformers =[
    (‘num_processing’, numerical_processing, ('feature1', 'feature3')),
    (‘cat_processing’, categorical_processing, ('feature0', 'feature2’))])

pipeline = Pipeline([(‘data_processing’, processor),
                     (‘estimator’, KNeighborsClassifier())])

pipeline.fit(X_train, y_train)
predictions = pipeline.predict(X_test)
```