Natural Language Processing

Encoding: Counting Encoding for Language

KyungTae Lim

Contents

Introduction of NLP

- NLP(Natural Language Processing, 자연어 처리)
  - A series of techniques that solve problems using statistical methods to understand text, regardless of linguistic knowledge.
    - 'Understanding' text is mainly achieved by transforming the text into computable representations
    - Representations can be discrete or continuous structures combined, such as vectors, tensors, graphs, trees
  - Products such as Alexa, Siri, and Google Translate are natural language processing applications.

Introduction of NLP related keywords

- Machine Learning
  - A technology that enables learning from data based on human-defined models and feature extraction methods and infers from them.
- Deep Learning
  - A machine learning technique that utilizes artificial neural networks.
  - Proven to be effective in natural language processing, speech, and computer vision
  - Deep Learning Frameworks
    - It provides various libraries and pre-trained deep learning algorithms, a kind of package
    - E.g.) TensorFlow, Keras, PyTorch

1.1 Supervised Learning

What kind of ML techniques are used in NLP area?

- Supervised Learning

  - A method of training using data that has known answers.

  - Labels are provided to the input when the input is given to the learning algorithm for training.

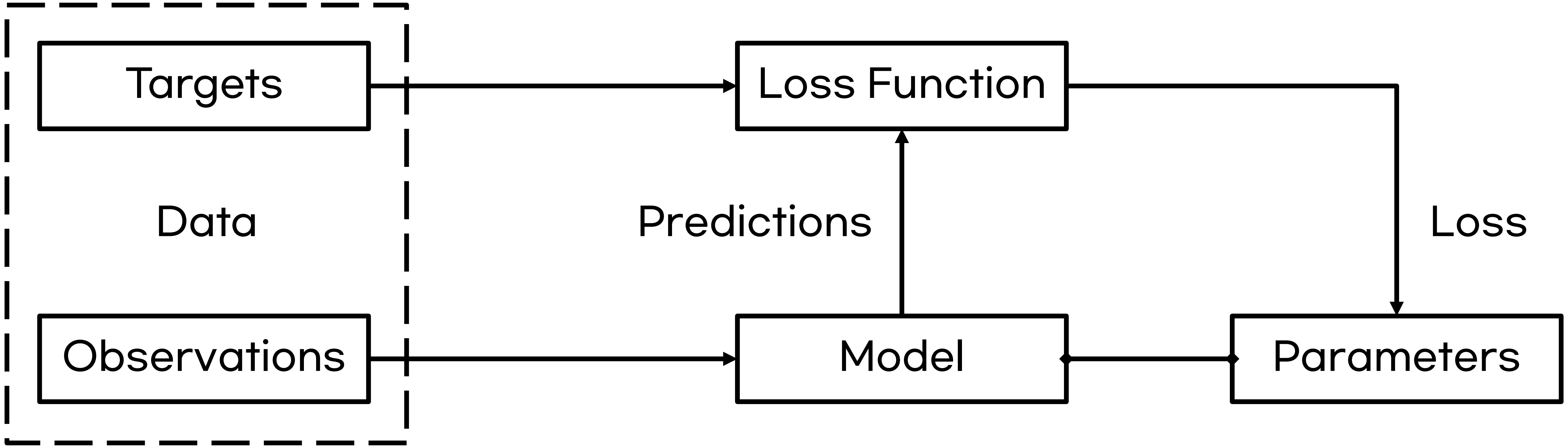
  - Classification and regression problems are typical examples.

- Unsupervised Learning

  - A method of predicting results for new data by clustering similar features in data without labels.

  - Typical examples include clustering and self-supervised learning

1.1 Supervised Learning



- Observations / sample

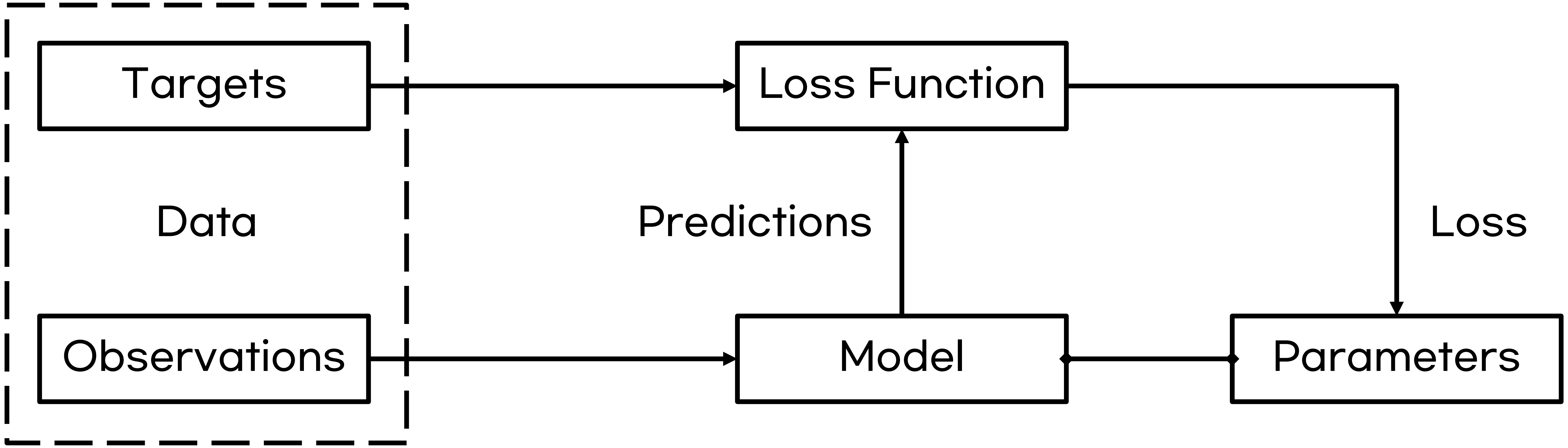
  - The item used for prediction is denoted as x and referred to as input.

- Targets

  - The target that is generally predicted with the label corresponding to the sample.

  - Denoted as y and referred to as the ground truth.

1.1 Supervised Learning



- Model

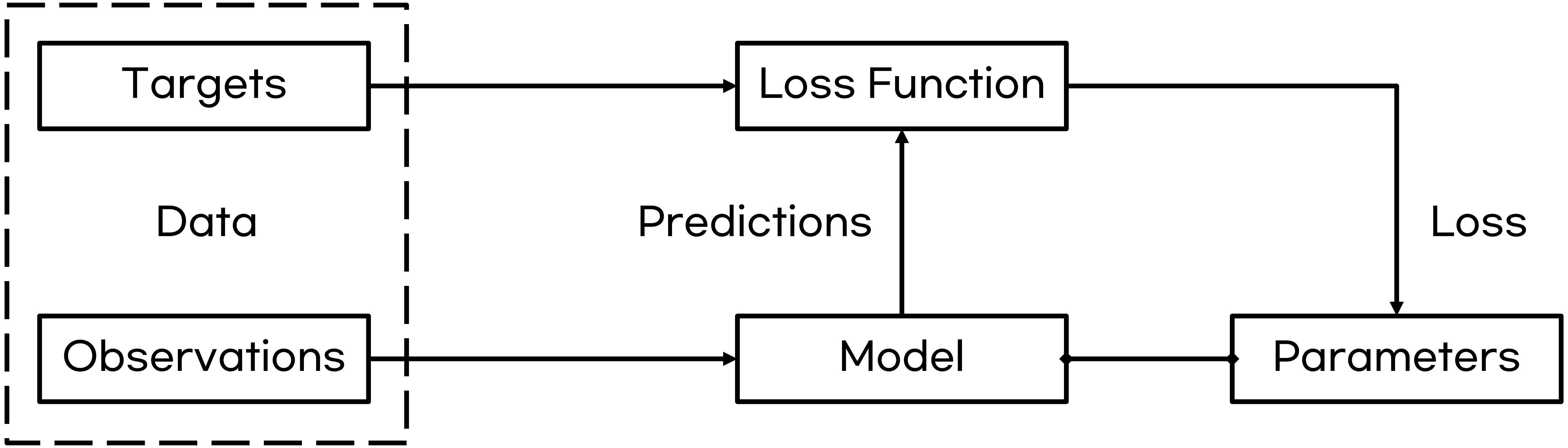
  - A function takes a mathematical expression and sample X and predicts its target

- Parameters

  - Also called weights, these are parameters that define the model.

  - Normally denoted w, $\hat{w}$

1.1 Supervised Learning



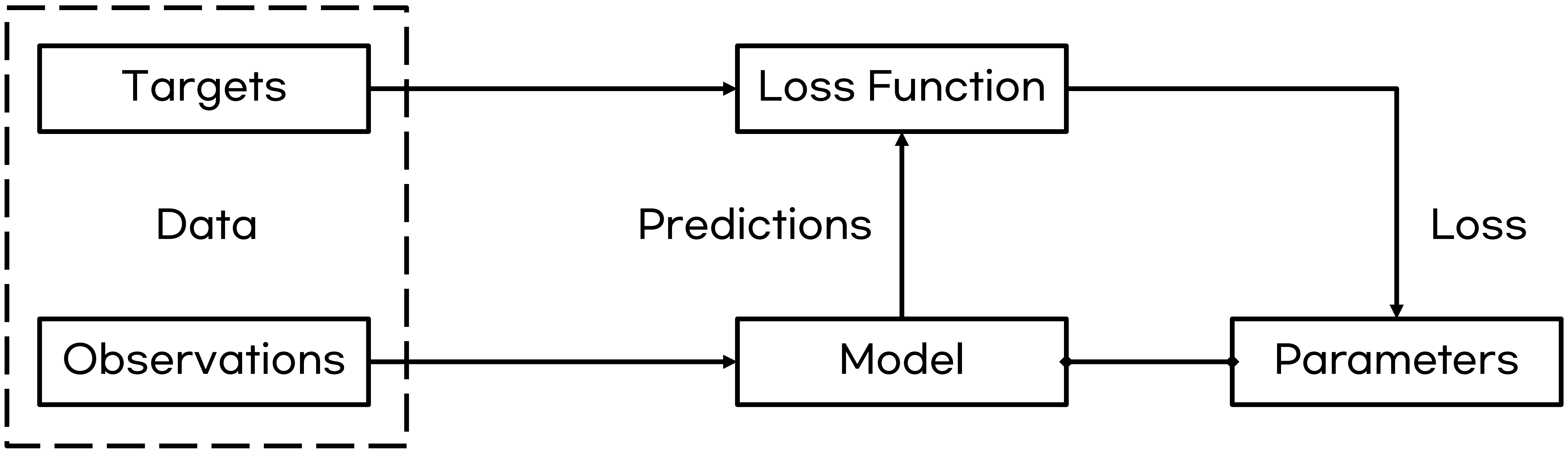
- Predictions

  - The target value estimated by the model, also referred to as an estimate.

  - It is represented using a hat symbol ($\hat{}$) over the variable,

    for example, the prediction of the target y is represented as $\hat{y}$.

1.1 Supervised Learning



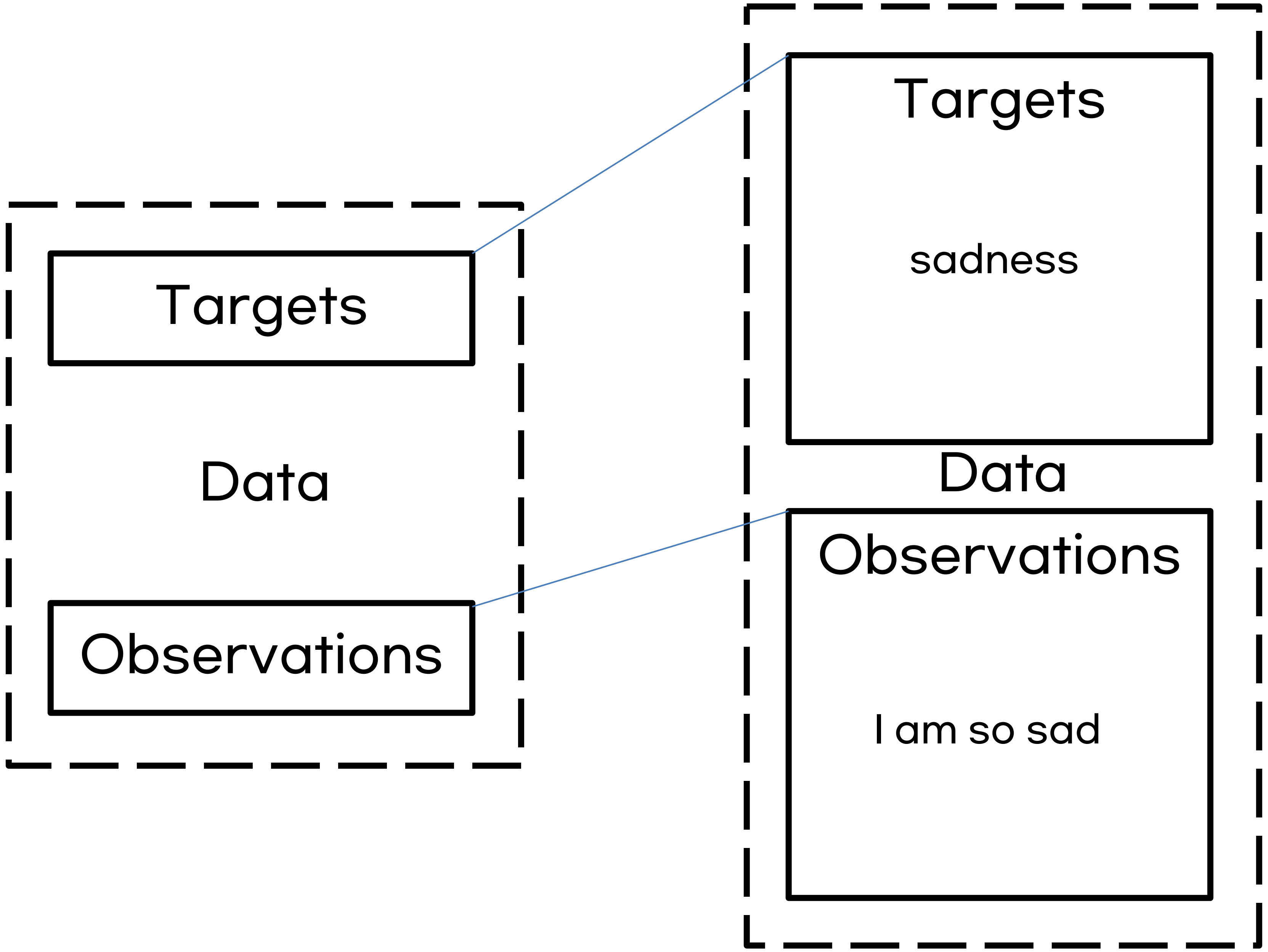
- Loss function

  - A function that compares how far the prediction for the training data is from the target.

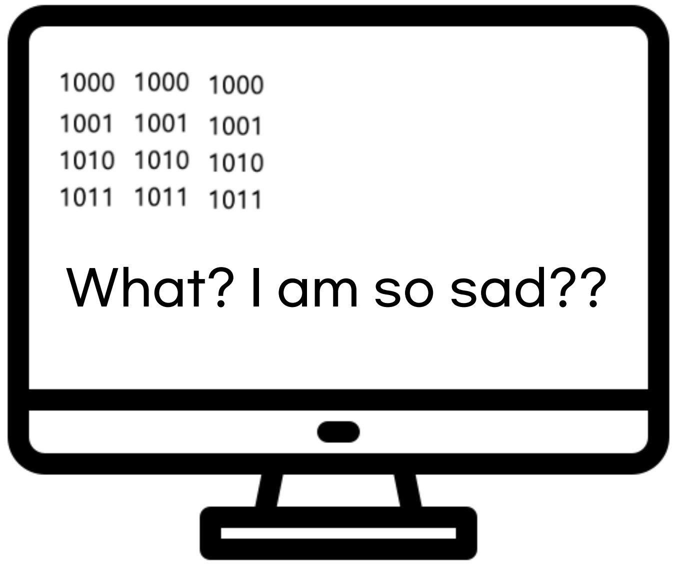
  - When given the target and prediction, it calculates a real-valued scalar called loss, where a lower loss indicates a better prediction by the model.

  - Denoted as L

1.2 Encoding



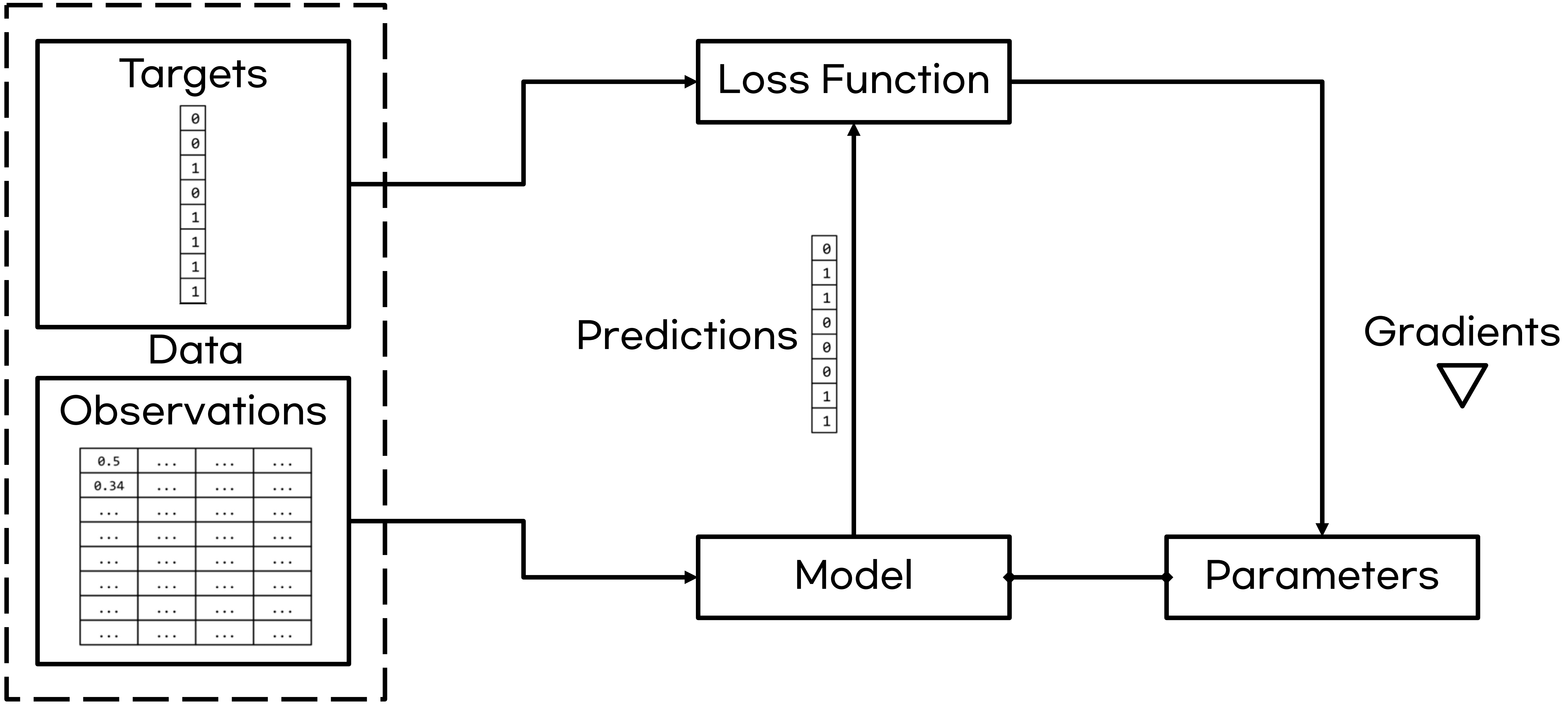
Hmmm, if the observation and targets are just text, how can the computer perform calculations?



1.2 Encoding

- Encoding

  - To use the sample (text) and target with machine learning algorithms, they need to be represented as numerical values in the form of vectors or tensors.



1.2 Encoding

- One-Hot Representation

  - Start with a zero vector and set the elements corresponding to the words in a sentence to 1

	time	fruit	flies	like	a	an	arrow	banana
1_{time}	1	0	0	0	0	0	0	0
1_{fruit}	0	1	0	0	0	0	0	0
1_{flies}	0	0	1	0	0	0	0	0
1_{like}	0	0	0	1	0	0	0	0
1_{a}	0	0	0	0	1	0	0	0
1_{an}	0	0	0	0	0	1	0	0
1_{arrow}	0	0	0	0	0	0	1	0
1_{banana}	0	0	0	0	0	0	0	1

- Let just assume we got two different sentences
- 1) Time flies like an arrow.
- 2) Fruit flies like a banana.
  - We can get 8 different vocabularies as a dictionary

    {time, fruit, flies, like, a, an, arrow, banana}

  → Each word can be represented as an 8-dimensional one-hot vector

  - Let just try to build one-hot encoding on the sentence

    "Time flies like an arrow"

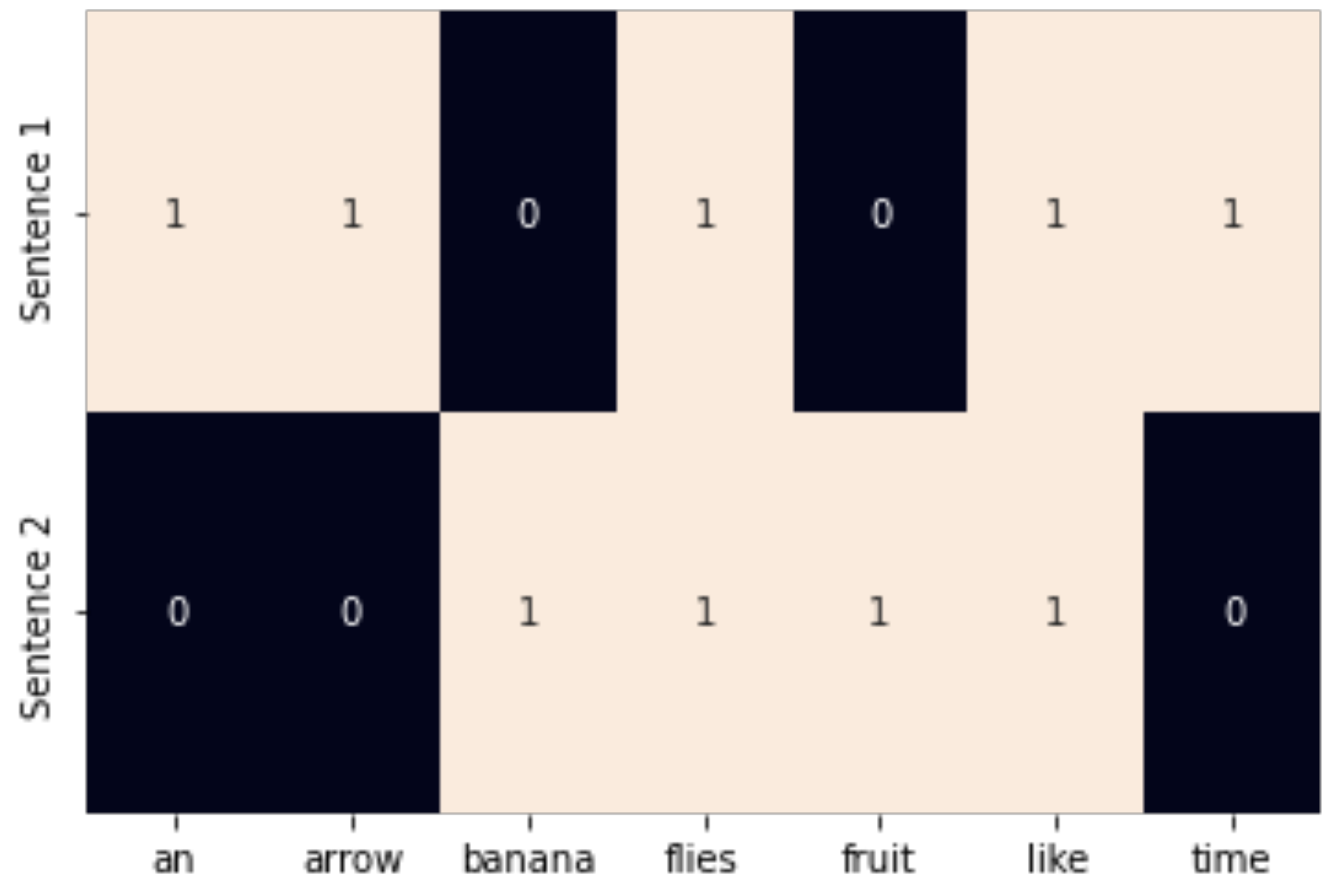
1.2 Encoding

- Codes for One-hot encoding

```
sentence = "Fruit flies like a banana"
tokens = sentence.split(sep=" ")

word_to_index = {word : index for index, word in enumerate(tokens)}
print('word dictionary :',word_to_index)

def one_hot_encoding(word, word_to_index):
    one_hot_vector = [0]*(len(word_to_index))
    index = word_to_index[word]
    one_hot_vector[index] = 1
return one_hot_vector
```

1.2 Encoding

```python
from sklearn.feature_extraction.text import CountVectorizer
import seaborn as sns

corpus = ['Time flies like an arrow.',
          'Fruit flies like a banana']
one_hot_vectorizer = CountVectorizer(binary=True)
one_hot = one_hot_vectorizer.fit_transform(corpus).toarray()
vocab = one_hot_vectorizer.get_feature_names()
sns.heatmap(one_hot, annot=True, cbar=False,
            xticklabels=vocab, yticklabels=['Sentence 1', 'Sentence 2'])
```



- Creating one-hot vectors or binary representations using scikit-learn
  - CountVectorizer(binary=True) : Set all non-zero counts to 1 for one-hot encoding, use True to use it for one-hot encoding
    - · The default value is False, which creates a TF representation that records the frequency of word occurrences
  - The CountVectorizer class ignores words consisting of a single character by default and does not include 'a'.

1.2 Encoding

Wait, why are we converting the strings into one-hot vectors?

Can't we just use integers instead, it's much easier?

- Let assume that we got four words {사과, 토끼, 토마토, 배}

  - We can set {사과=1, 토끼=2, 토마토=3, 배=4 } as discrete representation. Then calculate

  - Difference between Apple and Pear : |1-4| = 3

  - Difference between Apples and Rabbit: |1-2| = 1

- Differences in the similarity values of words occur depending on the order index

- In contrary, the deviation of one-hot encoding is both 2 (사과-배, 사과-토끼 모두 2)

1.2 Encoding

The word representation makes sense,

but then how do we represent a sentence?

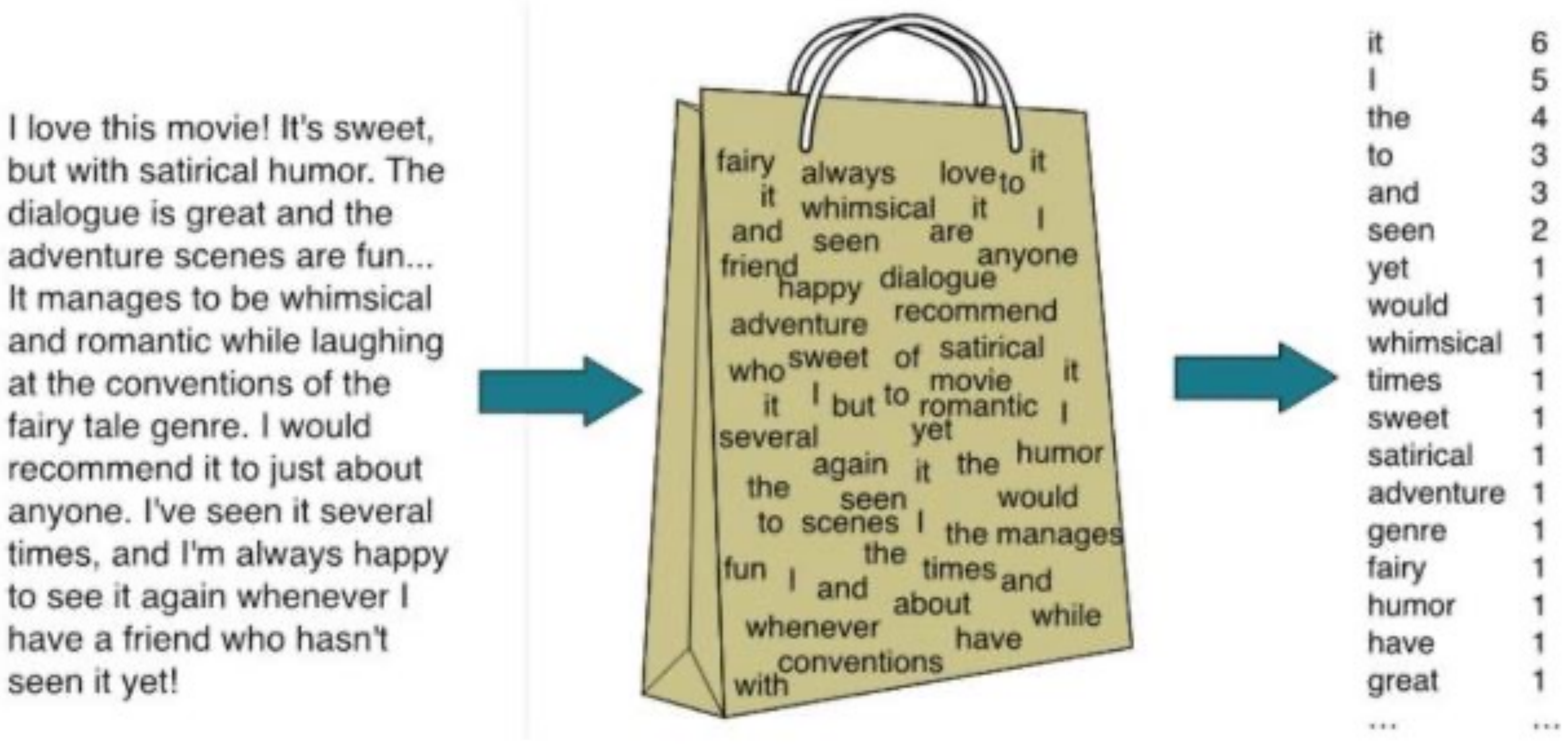
- Term-Frequency

  - The TF representation of a phrase, sentence, or document is the sum of the one-hot representations of words
  - The TF representation of 'Fruit flies like time flies a fruit' is [1, 2, 2, 1, 1, 0, 0, 0]
  - Each element represents the number of times the corresponding word appears in the sentence
    - In NLP, we call this "corpus"
    - This representation method is also called BoW (Bag of Words) model.
  - Denoted as $\mathrm{TF}(w)$ for a word (w)'s TF

1.2 Encoding

Bag of Words (BoW)??

- Bag of Words (BoW) is a numerical representation of text data that focuses only on the frequency of occurrence of words without considering the order of words.



https://slideplayer.com/slide/7073400/

1.2 Encoding

Coding Procedure of Bag of Words

- Creating a dictionary: Assign a unique integer index to each word.
- Creating vectors: to record the frequency of appearance of each word token at the index position.
- Let just build your own BoW using Naver Sentiment movie corpus v1.0

  - Download https://raw.githubusercontent.com/e9t/nsmc/master/ratings.txt

  - Comparing the vector values of “좋은데” and “별로인데”

Those who want English corpus. Get access to https://www.imdb.com/interfaces/

Reference https://raw.githubusercontent.com/e9t/nsmc/master/ratings.txt

1.2 Encoding

Implementation of BoW

```
from konlpy.tag import Okt

okt = Okt()

def build_bag_of_words(document):
# extract Molphologies
  document = document.replace('.', ”)
  tokenized_document = okt.morphs(document)

  word_to_index = {}
  bow = []

  for word in tokenized_document:
    if word not in word_to_index.keys():
      word_to_index[word] = len(word_to_index)
      # BoW에 전부 기본값 1을 넣는다.
      bow.insert(len(word_to_index) - 1, 1)
    else:
      # 재등장하는 단어의 인덱스
      index = word_to_index.get(word)
      # 재등장한 단어는 해당하는 인덱스의 위치에 1을 더한다.
      bow[index] = bow[index] + 1

return word_to_index, bow
```

```
doc1 = "정부가 발표하는 물가상승률과 소비자가 느끼는 물가상승률은 다르다."
vocab, bow = build_bag_of_words(doc1)
print('vocabulary :', vocab)
print('bag of words vector :', bow)

doc2 = '소비자는 주로 소비하는 상품을 기준으로 물가상승률을 느낀다.'

vocab, bow = build_bag_of_words(doc2)
print('vocabulary :', vocab)
print('bag of words vector :', bow)

doc3 = doc1 + ' ' + doc2
vocab, bow = build_bag_of_words(doc3)
print('vocabulary :', vocab)
print('bag of words vector :', bow)
```

BoW of Doc1 on Doc3 : [1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
BoW of Doc2 on Doc3 : [0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 2, 1, 1, 1]

1.2 Encoding

I understand the word and sentence representation now!!

Then what about the documentation?

- Document-Term Matrix (DTM)

  - A representation method that combines BoWs of different multiple documents.

Doc1 : 먹고 싶은 사과
Doc2 : 먹고 싶은 바나나
Doc3 : 길고 노란 바나나 바나나
Doc4 : 저는 과일이 좋아요

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
Doc1	0	0	0	1	0	1	1	0	0
Doc2	0	0	0	1	1	0	1	0	0
Doc3	0	1	1	0	2	0	0	0	0
Doc4	1	0	0	0	0	0	0	1	1

Reference https://wikidocs.net/24559

1.2 Encoding

Does Document-Term Matrix represent documents well?

- What is the most similar sentence to document 1 among the following documents?

Doc1 : the dog is so cute
Doc2 : this is the dog that I want to have
Doc3 : the day I am waiting for the lecture
Doc4 : you got the cute cat

- We consider other documents with many common words with Document 1 to be similar documents.

Does this mean that the "the" in Doc3 and the "the" in Doc1 make them similar?

Rare words are important!!!

1.2 Encoding

- Term-Frequency-Inverse-Document-Frequency (TF-IDF)

  - The value of multiplication between TF and IDF $\mathrm{TF(w)} \times \mathrm{IDF(w)}$

  - IDF(Inverse-Document-Frequency, 역문서 빈도)

    - Rare words don't appear frequently but can represent the characteristics of the document well.
    - In vector representation, the score of common tokens is lowered, and the score of rare tokens is increased.

  - $\mathrm{IDF(w)} = \log \frac{\mathrm{N}}{\mathrm{n_w}}$, $\mathrm{n_w}$: The number of documents that contain the word w. N : total # of Doc
  - A very common word ($\mathrm{n_w} = \mathrm{N}$) that appears in all documents has $\mathrm{IDF(w)} = 0$,

    and if it appears in only one document, the maximum value is $\mathrm{IDF(w)} = \log \mathrm{N}$

1.2 Encoding

- IDF(Inverse-Document-Frequency)

단어	IDF(역 문서 빈도)
과일이	ln(4/(1+1)) = 0.693147
길고	ln(4/(1+1)) = 0.693147
노란	ln(4/(1+1)) = 0.693147
먹고	ln(4/(2+1)) = 0.287682
바나나	ln(4/(2+1)) = 0.287682
사과	ln(4/(1+1)) = 0.693147
싶은	ln(4/(2+1)) = 0.287682
저는	ln(4/(1+1)) = 0.693147
좋아요	ln(4/(1+1)) = 0.693147

Doc1 : 먹고 싶은 사과
Doc2 : 먹고 싶은 바나나
Doc3 : 길고 노란 바나나 바나나
Doc4 : 저는 과일이 좋아요

$\mathrm{IDF}(w) = \log\frac{N}{n_w}$, n_w :
The number of documents that contain the word w.
N : total # of Doc

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
Doc1	0	0	0	1	0	1	1	0	0
Doc2	0	0	0	1	1	0	1	0	0
Doc3	0	1	1	0	2	0	0	0	0
Doc4	1	0	0	0	0	0	0	1	1

Reference https://wikidocs.net/24559

1.2 Encoding

Result of TF-IDF

- A value of multiplication between TF and IDF $TF(w) \times IDF(w)$

Doc1 : 먹고 싶은 사과
Doc2 : 먹고 싶은 바나나
Doc3 : 길고 노란 바나나 바나나
Doc4 : 저는 과일이 좋아요

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
Doc1	0	0	0	0.287682	0	0.693147	0.287682	0	0
Doc2	0	0	0	0.287682	0.287682	0	0.287682	0	0
Doc3	0	0.693147	0.693147	0	0.575364	0	0	0	0
Doc4	0.693147	0	0	0	0	0	0	0.693147	0.693147

Reference https://wikidocs.net/31698

1.2 Encoding

Coding Procedure of TF-IDF

- Creating a dictionary: Assign a unique integer index to each word.
- Creating vectors: to record the frequency of appearance of each word token at the index position.

  - tf(d,t) : The number of times a specific word t appears in a specific document d.

  - df(t) : The number of documents in which a specific word t appears.

  - idf(d, t) : The inverse proportion of df(t).

Let just build your own TF-IDF using Naver Sentiment movie corpus v1.0

- Download https://raw.githubusercontent.com/e9t/nsmc/master/ratings.txt

- Comparing the vector values of “좋은데” and “별로인데”

For English https://www.kaggle.com/datasets/dineshpiyasamara/sentiment-analysis-dataset

Reference https://raw.githubusercontent.com/e9t/nsmc/master/ratings.txt

1.2 Encoding

- Creating TF-IDF representation using Scikit-learn:

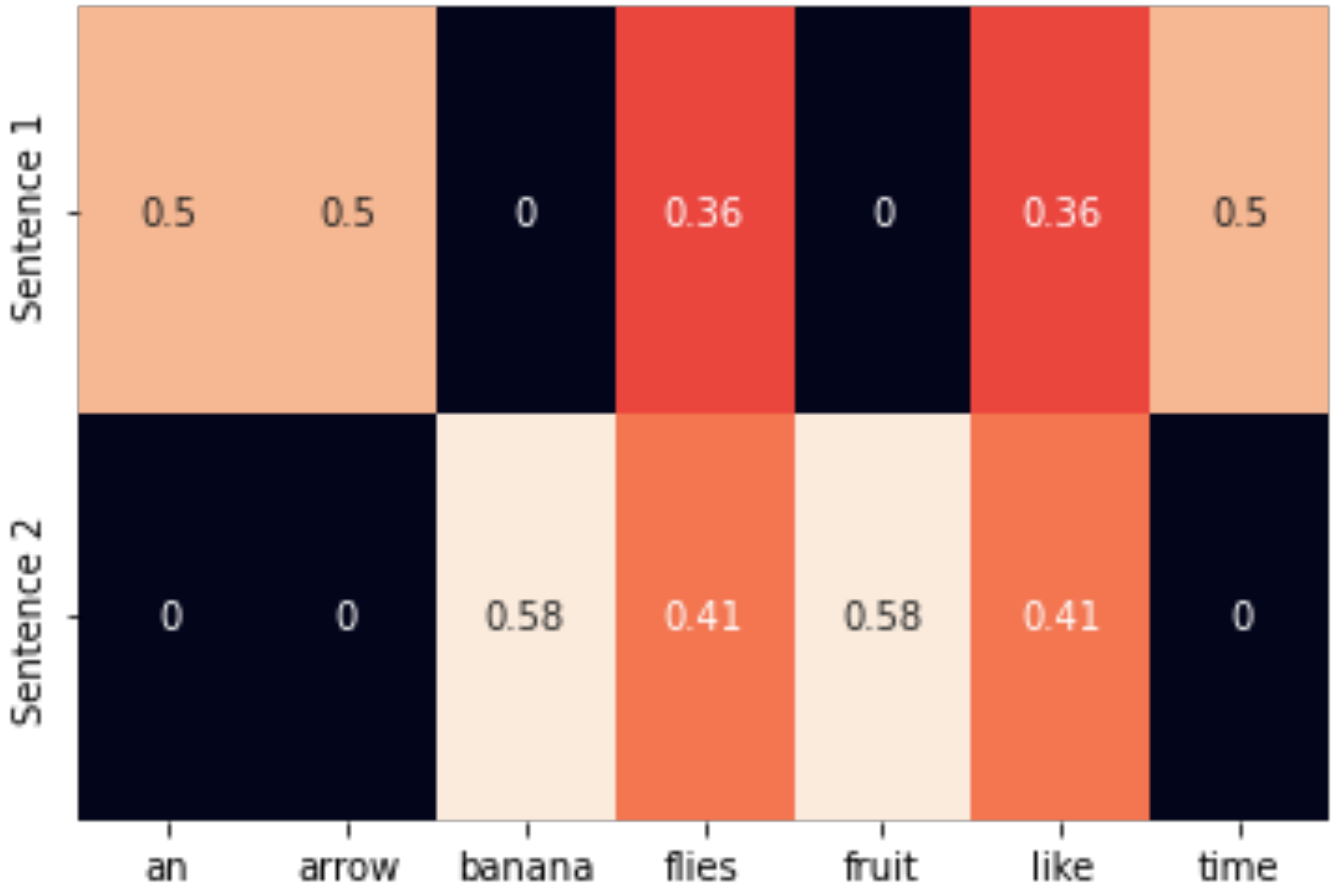
  - TfidfVectorizer

    · Add 1 to the numerator and denominator to prevent the denominator from becoming 0.

    Add 1 at the end to prevent IDF from being 0 when it is included in all documents.

    · $IDF(w) = \log(\frac{N+1}{n_w+1}) + 1$

```python
from sklearn.feature_extraction.text import TfidfVectorizer
import seaborn as sns

tfidf_vectorizer = TfidfVectorizer()
tfidf = tfidf_vectorizer.fit_transform(corpus).toarray()
sns.heatmap(tfidf, annot=True, cbar=False,
            xticklabels=vocab, yticklabels= ['Sentence 1', 'Sentence 2'])
```



1.2 Encoding

But what do you do with document vectors TF-IDF?

- If each document is expressed as a Vector,

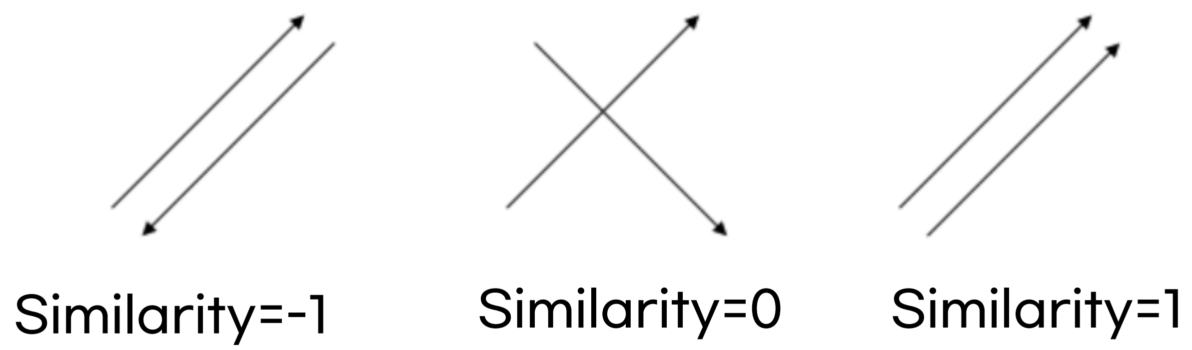
  the similarity of the Vector can be compared!

1.2 Encoding

Cosine Similarity

- It is a measure of similarity between two vectors using the cosine angle between them
- If the direction of the two vectors is exactly the same, it has a value of 1
- If they make a 90-degree angle, it has a value of 0,

and if they are in opposite directions with an angle of 180 degrees, it has a value of -1.



$$similarity = cos(\Theta) = \frac{A \cdot B}{||A||\ ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

1.2 Encoding

Implementation of Cosine Similarity

- Please code def cos_sim(A, B): function

  - A: a document (an np.array type vector)

  - B: another document (an np.array type vector)

$$similarity = cos(\Theta) = \frac{A \cdot B}{||A||\ ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

1.2 Encoding

Implementation of Cosine Similarity

```
doc1= "저는 사과 좋아요"
doc2= "저는 바나나 좋아요"
doc3= "저는 바나나 좋아요 저는 바나나 좋아요"

import numpy as np
from numpy import dot
from numpy.linalg import norm

def cos_sim(A, B):
return dot(A, B)/(norm(A)*norm(B))

doc1 = np.array([0,1,1,1])
doc2 = np.array([1,0,1,1])
doc3 = np.array([2,0,2,2])

print('similarity of Doc1 and Doc2:',cos_sim(doc1, doc2))
print('similarity of Doc1 and Doc3 :',cos_sim(doc1, doc3))
print('similarity of Doc2 and Doc3 :',cos_sim(doc2, doc3))
```

$$similarity = cos(\Theta) = \frac{A \cdot B}{||A||\ ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

Reference https://wikidocs.net/24603

1.2 Encoding

A movie recommendation Using TF-IDF and cosine similarity

- Recommend movies based on movie plots using TF-IDF and cosine similarity.

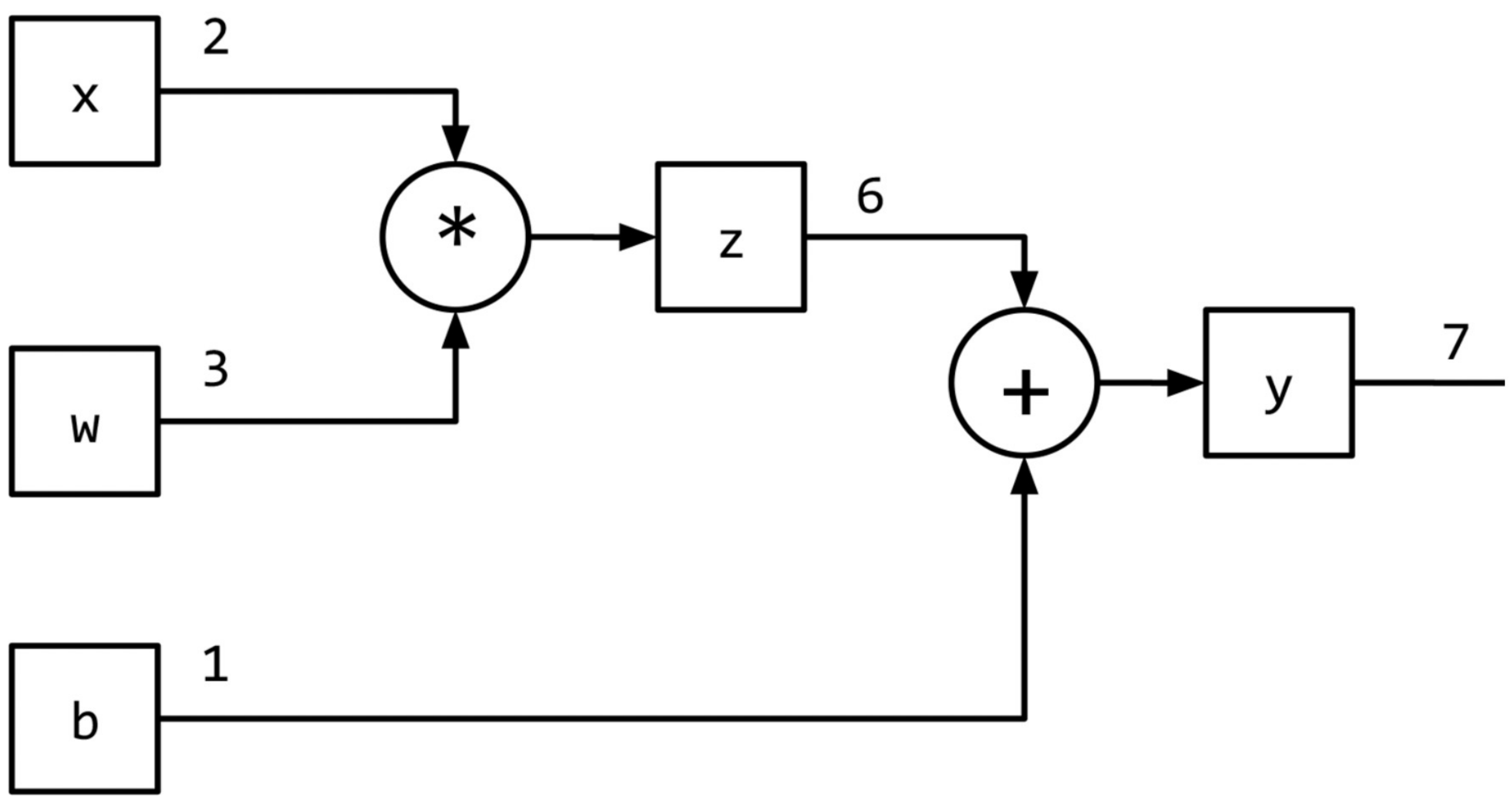
  - Data: https://www.kaggle.com/rounakbanik/the-movies-dataset
- 1) Read file
- 2) get the title and overview column
- 3) get rid of NULL values of the overview column
- 4) Compute tf-idf value over the overview column
- 5) Compute cosine-similarity (please check "from sklearn.metrics.pairwise import cosine_similarity")
- 6) Search a movie title from the data set and TOP 10 most similar ones

1.2 Encoding

- Target encoding
  - The exact form of the target variable depends on the NLP problem
  - In machine translation, summarization, and Q&A, the target is also text and is encoded in the same way as one-hot encoding.
  - Categorical labels
    - · Encoding in a way that gives each label a unique index
    - · Problematic when the number of output labels becomes too large

1.3 Computational Graph

- An abstract model of a mathematical expression
- Easily implement data flow
- In deep learning, implementations of computational graphs using TensorFlow, and Python with automatic differentiation. ex. $y = wx + b$



- Node : Math operations.
  - e.g., multiplication or addition

  Edge : Straight lines between nodes

Thank you