

Chapter_09_요약_few-to-no-labels.ipynb-text의 라벨을 tag하기위한 방법

transformer · 2024. 8. 30. 17:04

text의 라벨을 tag하기위한 방법

+ 코드 + 텍스트

stas00 commented on 1 Feb · edited · Description

Collaborator

Background

ZeRO-DP (ZeRO Data Parallel) and PP (Pipeline Parallelism) provide each a great memory saving over multiple GPUs. Each 1D allows for a much more efficient utilization of the gpu memory, but it's still not enough for very big models - sometimes not even feasible with any of the existing hardware. e.g. a model that's 45GB-big with just model params (t5-11b) can't fit even on a 40GB GPU.

The next stage in Model Parallelism that can enable loading bigger models onto smaller hardware is 2D Parallelism. That's combining Pipeline Parallelism (PP) with ZeRO-DP.

3D Parallelism is possible too and it requires adding a horizontal MP (ala [Megatron-LM](#), but we don't quite have any way to implement that yet. Need to study Megatron-LM first. So starting with a relatively low hanging fruit of 2D.

Tracking

We have 3 implementations that provide the required components to build 2D Parallelism:

1. DeepSpeed (DS)
2. FairScale (FS)
3. PyTorch (native) (PT)

and the purpose of this issue is to track the feasibility/status/inter-operability in each one of them. And also which parts have been back-ported to PyTorch core.

Plus it tracks the status of where transformers models are at with regards to the above 3 implementations.

The 2 main questions are:

1. native 2D: how do we integrate a native PP with native ZeRO-DP (sharded) (e.g. can fairscale PP work with fairscale ZeRO-DP)
2. inter-operability 2D: is there a chance one implementation of PP/ZeRO-DP could work with one or both others ZeRO-DP/PP (e.g. can fairscale PP work with DeepSpeed ZeRO-DP).

Assignees

stas00

Tags

Labels

DeepSpeed Model Parallel Pipeline Parallel WIP

Projects

None yet

Milestone

No milestone

Linked pull requests

Successfully merging a pull request may close this issue.

None yet

Notifications

Customize

Subscribe

You're not receiving notifications from this thread.

2 participants

```

+ 코드 + 텍스트
[6]
name: DeepSpeed,
color: 4034F7,
default: False,
description: ''}]

df_issues["labels"] = (df_issues["labels"]
                        .apply(lambda x: [meta["name"] for meta in x]))
df_issues[["labels"]].head()

labels
0 []
1 []
2 [DeepSpeed]
3 []
4 []

라벨의 이름만 가져옴

[ ] df_issues["labels"].apply(lambda x: len(x)).value_counts().to_frame().T

labels 0 1 2 3 4 5
count 6440 3057 305 100 25 3

[ ] df_counts = df_issues["labels"].explode().value_counts()
print(f"레이블 개수: {len(df_counts)}")
# 상위 8개 레이블을 출력합니다.
df_counts.to_frame().head(8).T # 라벨이름

```

한 글에 대해 label이 달린 갯수

```

+ 텍스트
1 []
2 [DeepSpeed]
3 []
4 []

df_issues["labels"].apply(lambda x: len(x)).value_counts().to_frame().T

labels 0 1 2 3 4 5
count 6440 3057 305 100 25 3

한 글에 대해 label이 달린 갯수

df_counts = df_issues["labels"].explode().value_counts()
print(f"레이블 개수: {len(df_counts)}")
# 상위 8개 레이블을 출력합니다.
df_counts.to_frame().head(8).T # 라벨이름

레이블 개수: 65
labels wontfix model card Core: Tokenization New model Core: Modeling Help wanted Good First Issue Usage
count 2284 649 106 98 64 52 50 46

label_map = {"Core: Tokenization": "tokenization",
              "New model": "new model",
              "Core: Modeling": "model training",
              "Usage": "usage",
              "Core: Pipeline": "pipeline",
              "TensorFlow": "tensorflow or tf",

```

```

+ 텍스트
print(f"레이블 개수: {len(df_counts)}")
# 상위 8개 레이블을 출력합니다.
df_counts.to_frame().head(8).T # 라벨이름

레이블 개수: 65
labels wontfix model card Core: Tokenization New model Core: Modeling Help wanted Good First Issue Usage
count 2284 649 106 98 64 52 50 46

label_map = {"Core: Tokenization": "tokenization",
             "New model": "new model",
             "Core: Modeling": "model training",
             "Usage": "usage",
             "Core: Pipeline": "pipeline",
             "TensorFlow": "tensorflow or tf",
             "PyTorch": "pytorch",
             "Examples": "examples",
             "Documentation": "documentation"}

def filter_labels(x):
    return [label_map[label] for label in x if label in label_map]

df_issues["labels"] = df_issues["labels"].apply(filter_labels)
all_labels = list(label_map.values())

df_counts = df_issues["labels"].explode().value_counts()
df_counts.to_frame().T

labels tokenization new model model training usage pipeline tensorflow or tf pytorch documentation examples
count 106 98 64 46 42 41 37 28 24

```

label (정답)이 달리게 적은상황. 조치가 필요함

```

코드 + 텍스트
11] labels tokenization new model model training usage pipeline tensorflow or tf pytorch documentation examples
count 106 98 64 46 42 41 37 28 24

df_issues["split"] = "unlabeled"
mask = df_issues["labels"].apply(lambda x: len(x) > 0)
df_issues.loc[mask, "split"] = "labeled"
df_issues["split"].value_counts().to_frame()

count
split
unlabeled 9489
labeled 441

for column in ["title", "body", "labels"]:
    print(f"{column}: {df_issues[column].iloc[26][:500]}\n")

title: Add new CANINE model

body: # 🌟 New model addition

## Model description

Google recently proposed a new **C**haracter **A**rchitecture with **N**o
tokenization **l**ean **N**eural **E**ncoders architecture (CANINE). Not only the
title is exciting:

> Pipelined NLP systems have largely been superseded by end-to-end neural
modeling, yet nearly all commonly-used models still require an explicit

```

title, body, label을 데이터에서 갖고옴

```
코드 + 텍스트

12] count
split
unlabeled 9489
labeled 441

for column in ["title", "body", "labels"]:
    print(f"{column}: {df_issues[column].iloc[26][:500]}\n")

title: Add new CANINE model
body: # 🌟 New model addition

## Model description

Google recently proposed a new **C**haracter **A**rchitecture with **N**o
tokenization **I**n **N**eural **E**ncoders architecture (CANINE). Not only the
title is exciting:

> Pipelined NLP systems have largely been superseded by end-to-end neural
modeling, yet nearly all commonly-used models still require an explicit
tokenization step. While recent tokenization approaches based on data-derived
subword lexicons are less brittle than manually en

labels: ['new model']

14] df_issues["text"] = (df_issues
    .apply(lambda x: x["title"] + "\n\n" + x["body"], axis=1))
```

#중복텍스트제거

```
코드 + 텍스트

13] Google recently proposed a new **C**haracter **A**rchitecture with **N**o
tokenization **I**n **N**eural **E**ncoders architecture (CANINE). Not only the
title is exciting:

> Pipelined NLP systems have largely been superseded by end-to-end neural
modeling, yet nearly all commonly-used models still require an explicit
tokenization step. While recent tokenization approaches based on data-derived
subword lexicons are less brittle than manually en

labels: ['new model']

14] df_issues["text"] = (df_issues
    .apply(lambda x: x["title"] + "\n\n" + x["body"], axis=1))

len_before = len(df_issues)
df_issues = df_issues.drop_duplicates(subset="text") #중복제거
print(f"삭제된 중복 이슈: {(len_before-len(df_issues))/len_before:.2%}")

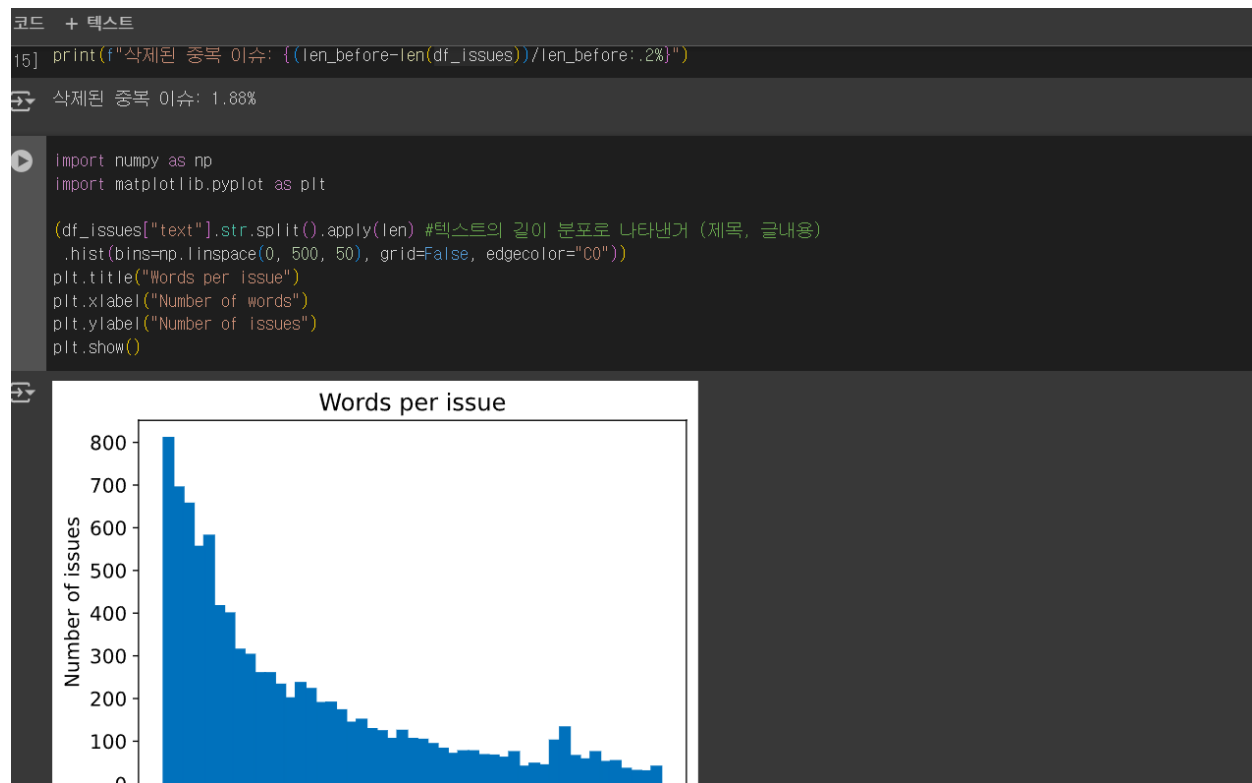
삭제된 중복 이슈: 1.88%

16] import numpy as np
import matplotlib.pyplot as plt

(df_issues["text"].str.split().apply(len)
    .hist(bins=np.linspace(0, 500, 50), grid=False, edgecolor="C0"))
plt.title("Words per issue")
plt.xlabel("Number of words")
plt.ylabel("Number of issues")
plt.show()
```

#텍스트의 길이

#텍스트의 길이 분포로 나타낸거 (제목, 글내용) 0.7 정도일때 성능이 제일 높았다가 threshold가 높으면
성능이 떨어짐



label(의 종류가 class) -> multi label classification


	class1	class2	class3
text1	0	0	1
text2	1	1	0
text3	1	1	1
text4	0	0	0
...			

아래그림

멀티라벨 - 글자를 숫자로 바꿔주

멀티라벨 - 글자를 숫자로 바꿔주는것

코드 + 텍스트



16]

훈련 세트 만들기

```

from sklearn.preprocessing import MultiLabelBinarizer

mlb = MultiLabelBinarizer() # 멀티라벨 - 글자를 숫자로 바꿔주는것
mlb.fit([all_labels])
mlb.transform([["tokenization", "new model"], ["pytorch"]])

```

array([[0, 0, 0, 1, 0, 0, 0, 1, 0],
 [0, 0, 0, 0, 0, 1, 0, 0, 0]])

```

] from skmultilearn.model_selection import iterative_train_test_split

def balanced_split(df, test_size=0.5):
    ind = np.expand_dims(np.arange(len(df)), axis=1)
    labels = mlb.transform(df["labels"])
    ind_train, _, ind_test, _ = iterative_train_test_split(ind, labels,
                                                            test_size)
    return df.iloc[ind_train[:, 0]], df.iloc[ind_test[:, 0]]

] from sklearn.model_selection import train_test_split

```

전체 라벨중에서 tokenization, new model label이 있는 곳이 1로 표시

코드 + 텍스트



18] all_labels

```

['tokenization',
 'new model',
 'model training',
 'usage',
 'pipeline',
 'tensorflow or tf',
 'pytorch',
 'examples',
 'documentation']

```

훈련 세트 만들기

```

from sklearn.preprocessing import MultiLabelBinarizer

mlb = MultiLabelBinarizer() # 멀티라벨 - 글자를 숫자로 바꿔주는것
mlb.fit([all_labels]) #라벨의 순서를 정해줌
mlb.transform([["tokenization", "new model"], ["pytorch"]])

```

array([[0, 0, 0, 1, 0, 0, 0, 1, 0],
 [0, 0, 0, 0, 0, 1, 0, 0, 0]])

전체 라벨중에서 tokenization, new model label이 있는 곳이 1로 표시

전체데이터를 train 데이터랑 test 데이터(validation)로 나누고있음

```
코드 + 텍스트

mlb.fit([all_labels]) #라벨의 순서를 정해줌
mlb.transform([["tokenization", "new model"], ["pytorch"]])

array([[0, 0, 0, 1, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 0]])

from sklearn.model_selection import iterative_train_test_split

def balanced_split(df, test_size=0.5): #전체데이터를 train 데이터랑 test 데이터(validation)로 나누고있음
    ind = np.expand_dims(np.arange(len(df)), axis=1)
    labels = mlb.transform(df["labels"])
    ind_train, _, ind_test, _ = iterative_train_test_split(ind, labels,
                                                            test_size)
    return df.iloc[ind_train[:, 0]], df.iloc[ind_test[:, 0]]

] from sklearn.model_selection import train_test_split

df_clean = df_issues[["text", "labels", "split"]].reset_index(drop=True).copy()
df_unsup = df_clean.loc[df_clean["split"] == "unlabeled", ["text", "labels"]]
df_sup = df_clean.loc[df_clean["split"] == "labeled", ["text", "labels"]]

np.random.seed(0)
df_train, df_tmp = balanced_split(df_sup, test_size=0.5)
df_valid, df_test = balanced_split(df_tmp, test_size=0.5)

] from datasets import Dataset, DatasetDict

ds = DatasetDict({
    "train": Dataset.from_pandas(df_train.reset_index(drop=True)),
    "valid": Dataset.from_pandas(df_valid.reset_index(drop=True)),
```

```
df_clean = df_issues[["text", "labels",
"split"]].reset_index(drop=True).copy() #원래데이터(중복제거한거를
0,1,2,3... 해서 clean됨)
df_unsup = df_clean.loc[df_clean["split"] == "unlabeled", ["text",
"labels"]] #라벨안달린 데이터를 df_unsup 즉 unsupervised에 넣고
df_sup = df_clean.loc[df_clean["split"] == "labeled", ["text",
"labels"]] #라벨달린 데이터를 df_sup supervised에 넣고,

np.random.seed(0)
df_train, df_tmp = balanced_split(df_sup, test_size=0.5) #tmp에 50%
담아둬 #라벨달린 df_sup을 split해서 train, tmp로 각각 50%로 나누고
df_valid, df_test = balanced_split(df_tmp, test_size=0.5) #다시 tmp를
valid, test로 50%씩 나눈다.
```

```
from datasets import Dataset, DatasetDict #datasets안에 Dataset,
DatasetDict를 갖고온다.
```

```
ds = DatasetDict({ #처음 코드에서 import pandas as pd #pandas안에
dataframe이라는 class가 있음 , dataframe은 표로 나와있음 . 이것을 dataset으로
```

바꿔줌. dataframe표는 다루기어려워서

```
"train": Dataset.from_pandas(df_train.reset_index(drop=True)),
"valid": Dataset.from_pandas(df_valid.reset_index(drop=True)),
"test": Dataset.from_pandas(df_test.reset_index(drop=True)),
"unsup": Dataset.from_pandas(df_unsup.reset_index(drop=True)))
```

상자

빨간공 95개 => 19개

파란공이 5개 => 1개

19:1 = 20개

공을 20개 뽑음 => 빨강 19개, 파랑 1개

훈련 슬라이스 만들기

```
np.random.seed(0)
all_indices = np.expand_dims(list(range(len(ds["train"]))), axis=1)
indices_pool = all_indices
labels = mlb.transform(ds["train"]["labels"])
train_samples = [8, 16, 32, 64, 128]
train_slices, last_k = [], 0

for i, k in enumerate(train_samples): # 라벨 비율을 맞춰서 [8, 16, 32, 64, 128] 뽑는다.
    # 다음 슬라이스 크기를 채우는데 필요한 샘플을 분할합니다
    indices_pool, labels, new_slice, _ = iterative_train_test_split(
        indices_pool, labels, (k-last_k)/len(labels))
    last_k = k
    if i==0: train_slices.append(new_slice)
    else: train_slices.append(np.concatenate((train_slices[-1],
new_slice)))

# 마지막 슬라이스를 포함하면 코랩의 경우 메모리 부족이 발생합니다.
# 대신 코랩 프로(https://colab.research.google.com/signup)를 사용하세요.
# 코랩을 사용하려면 다음 라인을 주석 처리하세요.
train_slices.append(all_indices),
train_samples.append(len(ds["train"])) #train_slices에 뽑아놓은거를 담아
```


둔다.

```
train_slices = [np.squeeze(train_slice) for train_slice in  
train_slices]
```

나이프 베이즈 모델 만들기

```
def prepare_labels(batch): #베이스 모델을 만들고 기준으로 삼는다. 라벨 텍스트를  
숫자로 매핑하는 코드  
    batch["label_ids"] = mlb.transform(batch["labels"])  
    return batch  
  
ds = ds.map(prepare_labels, batched=True)
```

naive bayes 도 학습을 한다. (베이스라인- 적당히 성능이 높지않은 기
준으로 삼을수 있는모델)

데이터를 counter vectorizer로 단어갯수별 행렬로 만듦

문서 단어 행렬(Document-Term Matrix)

CountVectorizer는 문서에서 단어의 빈도수를 계산해서 문서 단어 행렬을 만들어주는 작업을 하는 모듈입니다.

그러므로 우선 문서 단어 행렬이 무엇인지 알아보겠습니다.

분석 대상으로 삼는 문서가 다음과 같이 2개 있다고 예를 들어보겠습니다.

```
doc1 = "She likes python"
doc2 = "She hates python"
```

이 두 문서에 대해서 문서 단어 행렬을 만들면 다음과 같이 됩니다.

	She	likes	hate	python
doc1	1	1	0	1
doc2	1	0	1	1

말하자면, 각각의 문서에서 각각의 단어가 몇 번 등장했는지를 세서 행렬 형태로 표현해주는 것입니다.

이렇게 하면 doc1은 [1, 1, 0, 1]이라는 벡터로 표현되기 때문에 문서 간에 수치적인 연산이 가능하게 됩니다.

이번에는 다음 예를 보겠습니다.

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
from sklearn.feature_extraction.text import CountVectorizer

for train_slice in train_slices:
    # 훈련 슬라이스와 테스트 데이터를 준비합니다
    ds_train_sample = ds["train"].select(train_slice)
    y_train = np.array(ds_train_sample["label_ids"])
    y_test = np.array(ds["test"]["label_ids"])
    # 간단한 CountVectorizer를 사용해 텍스트를 토큰 카운트로 인코딩합니다
    count_vect = CountVectorizer() #어떤문장에 어떤단어가 있는지 벡터화한다.
    X_train_counts =
count_vect.fit_transform(ds_train_sample["text"])
    X_test_counts = count_vect.transform(ds["test"]["text"])
    # 모델을 만들고 훈련합니다!
    classifier = BinaryRelevance(classifier=MultinomialNB()) # 모델을
만드는 코드
    classifier.fit(X_train_counts, y_train) # train 데이터를 넣어서 학습
    # 예측을 생성하고 평가합니다
    y_pred_test = classifier.predict(X_test_counts) #테스트 데이터를 넣어
```

서 예측값을 뽑음

```
clf_report = classification_report( # 결과값을 갖고옴
    y_test, y_pred_test, target_names=mlb.classes_,
    zero_division=0,
    output_dict=True)
# 평가 결과를 저장합니다
macro_scores["Naive Bayes"].append(clf_report["macro avg"]["f1-
score"]) #macro_scores는 class별로 f1 score 를 매긴후 평균을 낸것
micro_scores["Naive Bayes"].append(clf_report["micro avg"]["f1-
score"]) #전체 데이터에 대해서 평균을 낸것 -> 정확도
```

레이블이 없는 데이터 다루기

제로샷 런닝

이미 학습이 잘된 모델을 갖고 데이터를 넣어보는것(파라미터수정안함)

```
from transformers import pipeline
```

```
pipe = pipeline("fill-mask", model="bert-base-uncased") # fill-mask불
러와서 mask 에 뭐가 들어갈지 결정할것
```

```
movie_desc = "The main characters of the movie madagascar \
are a lion, a zebra, a giraffe, and a hippo. "
prompt = "The movie is about [MASK]." #MASK에 정답으로 animal이 제일 높
게 나왔음
```

```
output = pipe(movie_desc + prompt)
for element in output:
    print(f"토큰 {element['token_str']}:\t{element['score']:.3f}%")
```

```
토큰 animals:      0.103%
토큰 lions:         0.066%
토큰 birds:         0.025%
```

```
토큰 love:      0.015%
토큰 hunting:    0.013%
```

```
output = pipe(movie_desc + prompt, targets=["animals", "cars"]) #
animal과 cars중 위쪽 movie_desc에 어떤값이 더 적합한지 물어봄
for element in output:
    print(f"토큰 {element['token_str']}:\t{element['score']:.3f}%")
```

```
토큰 animals:    0.103%
토큰 cars:       0.001%
```

```
from transformers import pipeline

# CPU에서 실행하려면 device=0을 삭제하세요.
pipe = pipeline("zero-shot-classification", device=0) #pipe line으로
zeroshot을 불러옴
```

```
sample #sample 변수에 #train 데이터에서 0번째(첫번째) 데이터를 갖고옴 #
```

```
{'text': "Add new CANINE model\n\n# 🌟 New model addition\n\n\n\n##
Model description\n\n\n\nGoogle recently proposed a new
**C**haracter **A**rchitecture with **N**o tokenization **I**n
**N**eural **E**ncoders architecture (CANINE). Not only the title is
exciting:\n\n\n\n> Pipelined NLP systems have largely been
superseded by end-to-end neural modeling, yet nearly all commonly-
used models still require an explicit tokenization step. While
recent tokenization approaches based on data-derived subword
lexicons are less brittle than manually engineered tokenizers, these
techniques are not equally suited to all languages, and the use of
any fixed vocabulary may limit a model's ability to adapt. In this
```

paper, we present CANINE, a neural encoder that operates directly on character sequences, without explicit tokenization or vocabulary, and a pre-training strategy that operates either directly on characters or optionally uses subwords as a soft inductive bias. To use its finer-grained input effectively and efficiently, CANINE combines downsampling, which reduces the input sequence length, with a deep transformer stack, which encodes context. CANINE outperforms a comparable mBERT model by 2.8 F1 on TyDi QA, a challenging multilingual benchmark, despite having 28% fewer model parameters.

Overview of the architecture:

[outputname-1](https://user-images.githubusercontent.com/20651387/113306475-6c3cac80-9304-11eb-9bad-ad6323904632.png)

Paper is available [here] (https://arxiv.org/abs/2103.06874).

We heavily need this architecture in Transformers (RIP subword tokenization)!

The first author (Jonathan Clark) said on [Twitter] (https://twitter.com/JonClarkSeattle/status/1377505048029134856) that the model and code will be released in April :partying_face:

Open source status

* [] the model implementation is available: soon [here](https://caninemodel.page.link/code)

* [] the model weights are available: soon [here] (https://caninemodel.page.link/code)

* [x] who are the authors: @jhclark-google (not sure), @dhgarrette, @jwieting (not sure)

"

```
'labels': ['new model'],
'label_ids': [0, 0, 0, 1, 0, 0, 0, 0, 0]}
```

all_labels

```
['tokenization',
 'new model',
 'model training',
 'usage',
 'pipeline',
 'tensorflow or tf',
 'pytorch',
```

```
'examples',  
'documentation']
```

```
sample = ds["train"][0] #train 데이터에서 0번째(첫번째) 데이터를 갖고와서  
sample변수에 넣어놓음 #  
print(f"레이블: {sample['labels']}")  
output = pipe(sample["text"], all_labels, multi_label=True) # 첫번째  
데이터중 all label 갖고와서 이중 멀티 라벨로 달릴수 있다.  
print(output["sequence"][:400])  
print("\n예측:")  
  
for label, score in zip(output["labels"], output["scores"]):  
    print(f"{label}, {score:.2f}")
```

```
레이블: ['new model']  
Add new CANINE model
```

```
# 🌟 New model addition
```

```
## Model description
```

```
Google recently proposed a new **C**haracter **A**rchitecture with  
**N**o
```

```
tokenization **I**n **N**eural **E**ncoders architecture (CANINE).
```

```
Not only the
```

```
title is exciting:
```

```
> Pipelined NLP systems have largely been superseded by end-to-end  
neural
```

```
modeling, yet nearly all commonly-used models still require an  
explicit tokeni
```

```
예측:
```

```
new model, 0.98
```

```
tensorflow or tf, 0.37
```

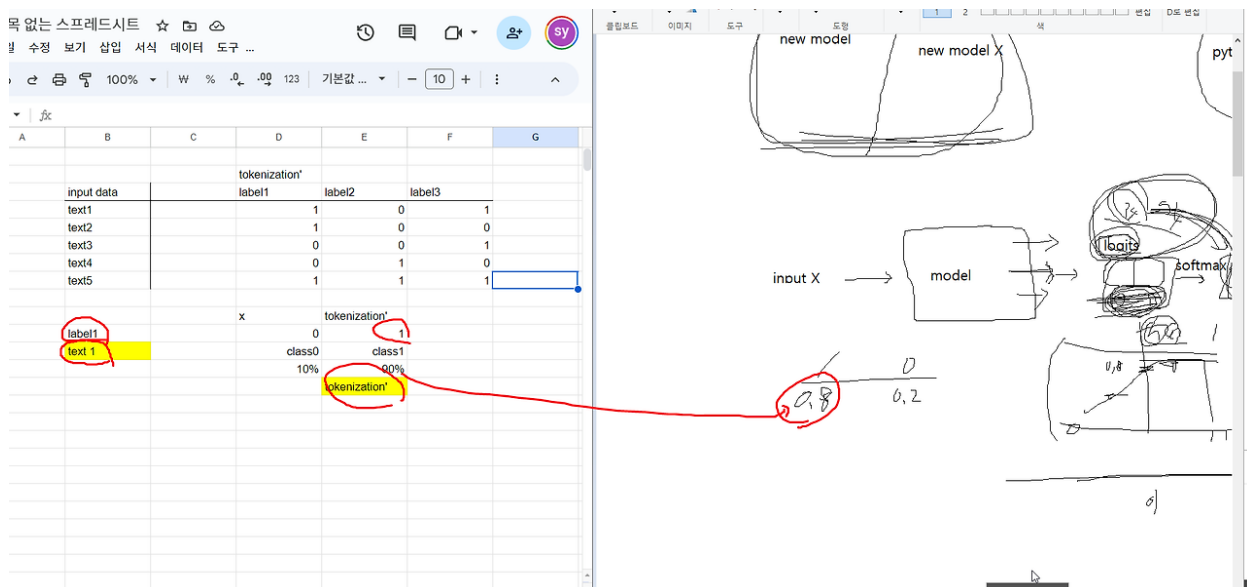
```
examples, 0.34
```

usage, 0.30
pytorch, 0.25
documentation, 0.25
model training, 0.24
tokenization, 0.17
pipeline, 0.16

아래그림) 멀티라벨이라서 전체라벨중 퍼센트

new model, 0.98
tensorflow or tf, 0.37
examples, 0.34
usage, 0.30
pytorch, 0.25
documentation, 0.25
model training, 0.24
tokenization, 0.17
pipeline, 0.16

이진분류로 각 text에 라벨이 들어있는지 확률값으로 구한다. 0이면 그라벨이 안들어있고, 1이면 들어있고,
확률값으로 80%이면 1로 치고 20%면 0으로 친다.



```
def zero_shot_pipeline(example):
    output = pipe(example["text"], all_labels, multi_label=True)
    example["predicted_labels"] = output["labels"]
    example["scores"] = output["scores"]
    return example

ds_zero_shot = ds["valid"].map(zero_shot_pipeline) #validation 데이터에 대해 각 text가 라벨이 어떻게 들어있는지 이진분류로 확인
```

아래의 첫번째 text를 sample에 넣은것의 결과값인

```
sample = ds["train"][0] #train 데이터에서 0번째(첫번째) 데이터를 갖고와서 sample변수에 넣어놓음 #
print(f"레이블: {sample['labels']}")
output = pipe(sample["text"], all_labels, multi_label=True) # 첫번째 데이터중 all label 갖고와서 이중 멀티 라벨로 달릴수 있다.
print(output["sequence"][:400])
print("\n예측:")

for label, score in zip(output["labels"], output["scores"]):
    print(f"{label}, {score:.2f}")
```

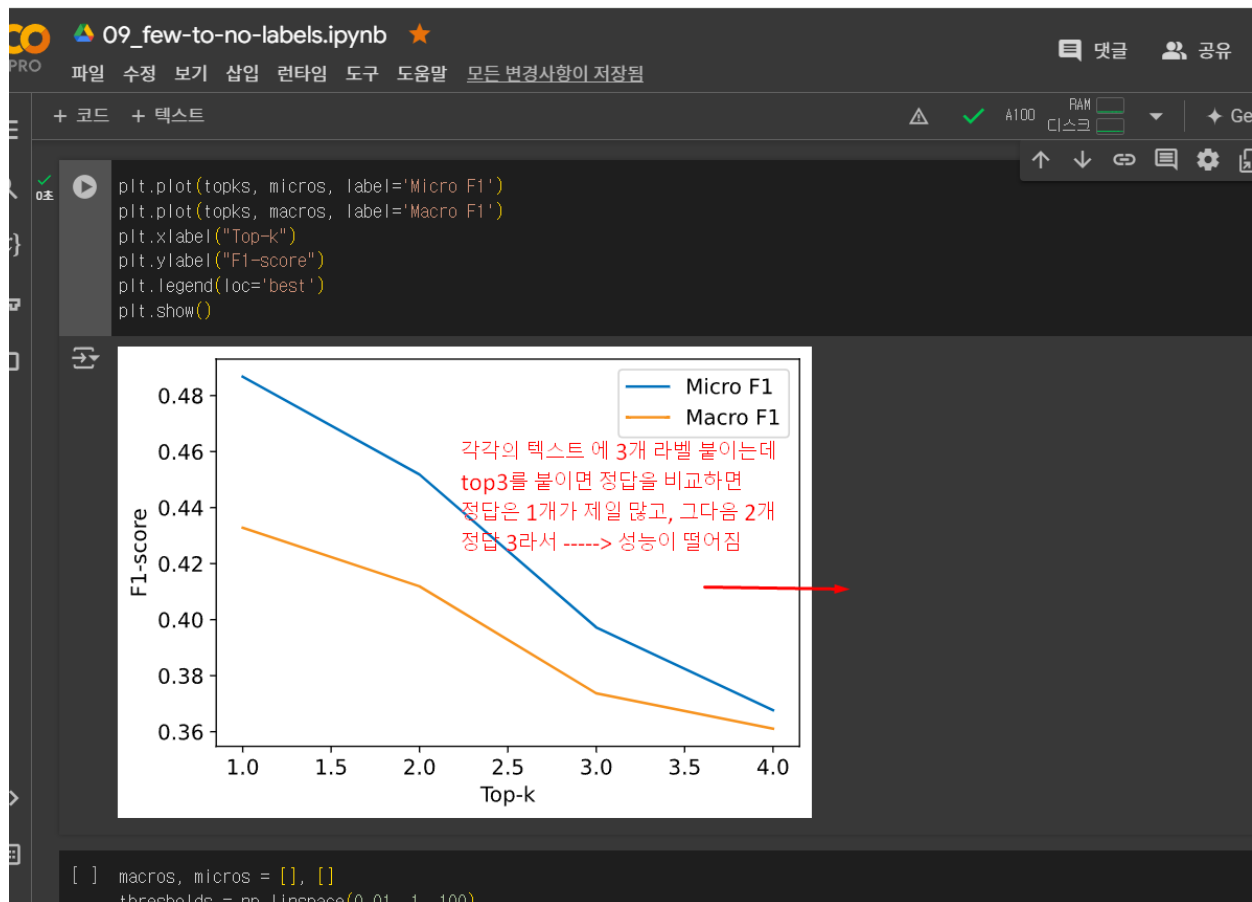
아래의 첫번째 text를 sample에 넣은것의 결과값인 아래 결과값중 topk 로 label을 잘라서 label 붙일것인가. 아님 확률값으로 잘라서 라벨을 붙일것인가 결정

```
예측:
new model, 0.98
tensorflow or tf, 0.37
examples, 0.34
usage, 0.30
pytorch, 0.25
documentation, 0.25
model training, 0.24
tokenization, 0.17
pipeline, 0.16
```


아래의 첫번째 text를 sample에 넣은것의 결과값인 아래 결과값중 topk 로 label을 잘라서 label 붙일것인가. 아님 확률값으로 잘라서 라벨을 붙일것인가 결정

```
def get_preds(example, threshold=None, topk=None):
    preds = []
    if threshold: #오른쪽 확률값의 몇프로 이상이면 라벨값으로 하겠다.
        for label, score in zip(example["predicted_labels"],
                                example["scores"]):
            if score >= threshold:
                preds.append(label)
    elif topk:
        for i in range(topk):
            preds.append(example["predicted_labels"][i])
    else:
        raise ValueError("`threshold` 또는 `topk`로 지정해야 합니다.")
    return {"pred_label_ids":
            list(np.squeeze(mlb.transform([preds]))]}
```

각각의 텍스트 에 3개 라벨 붙이는데
top3를 붙이면 정답을 비교하면
정답은 1개가 제일 많고, 그다음 2개
정답 3라서 -----> 성능이 떨어짐



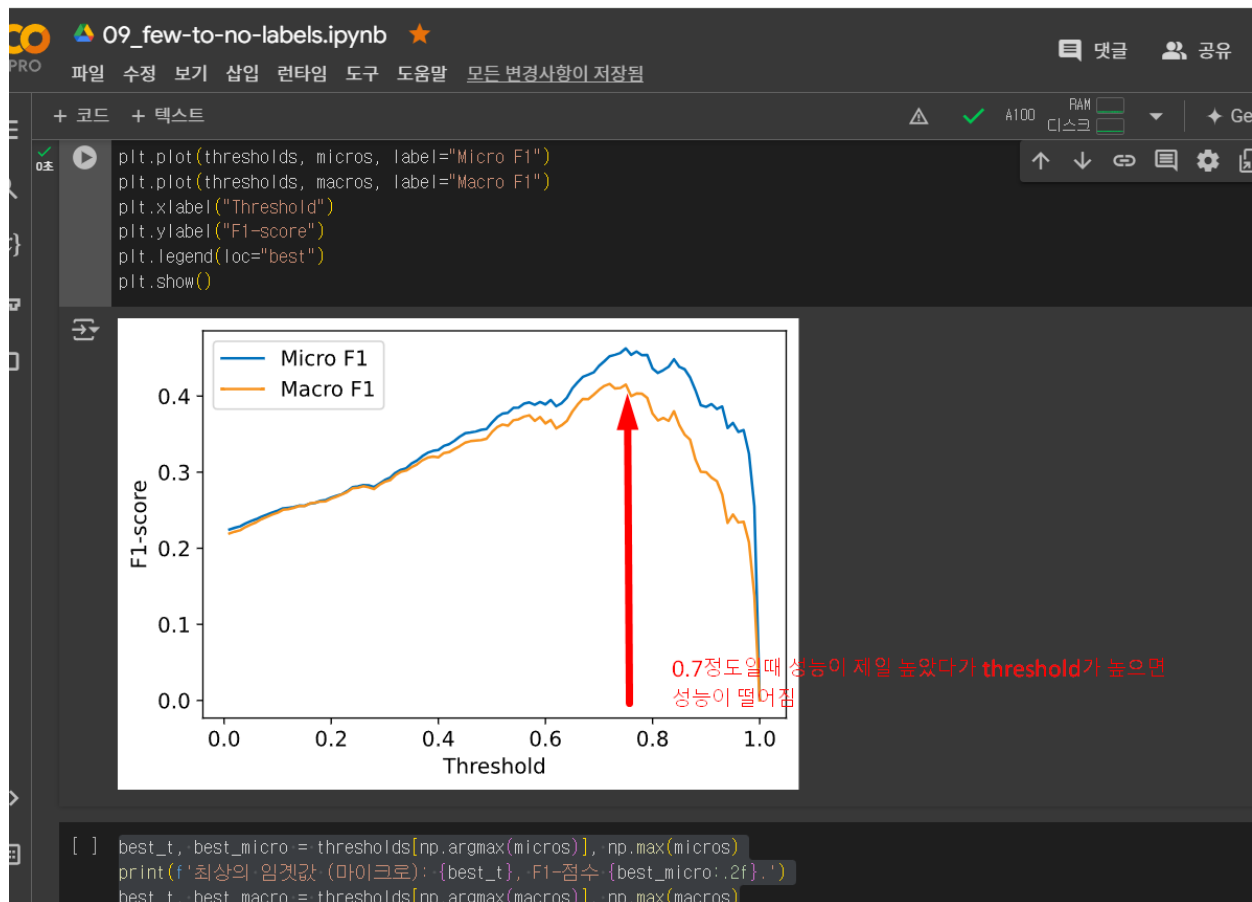
threshold로 0.01씩 올려서 라벨의 확률값을 올려봄

```

macros, micros = [], []
thresholds = np.linspace(0.01, 1, 100)
for threshold in thresholds:
    ds_zero_shot = ds_zero_shot.map(get_preds,
                                     fn_kwargs={"threshold":
threshold})
    clf_report = get_clf_report(ds_zero_shot)
    micros.append(clf_report["micro avg"]["f1-score"])
    macros.append(clf_report["macro avg"]["f1-score"])

```

0.7정도일때 성능이 제일 높았다가 threshold가 높으면
성능이 떨어짐



데이터 증식 으로 성능을 높임

but, 이미지는 데이터 증식이 상관없는데

자연어에서는 변형으로 데이터 손실이 있을수 있음

그래서 동의어로 변경하거나, 강아지 랑 개 --> 이렇게 비슷한것으로 변경 - 이렇게 증식
약간 이상하더라도 단어를 없애거나 추가함.

라벨

강아지 사진1 강아지

강아지 사진1-2 강아지

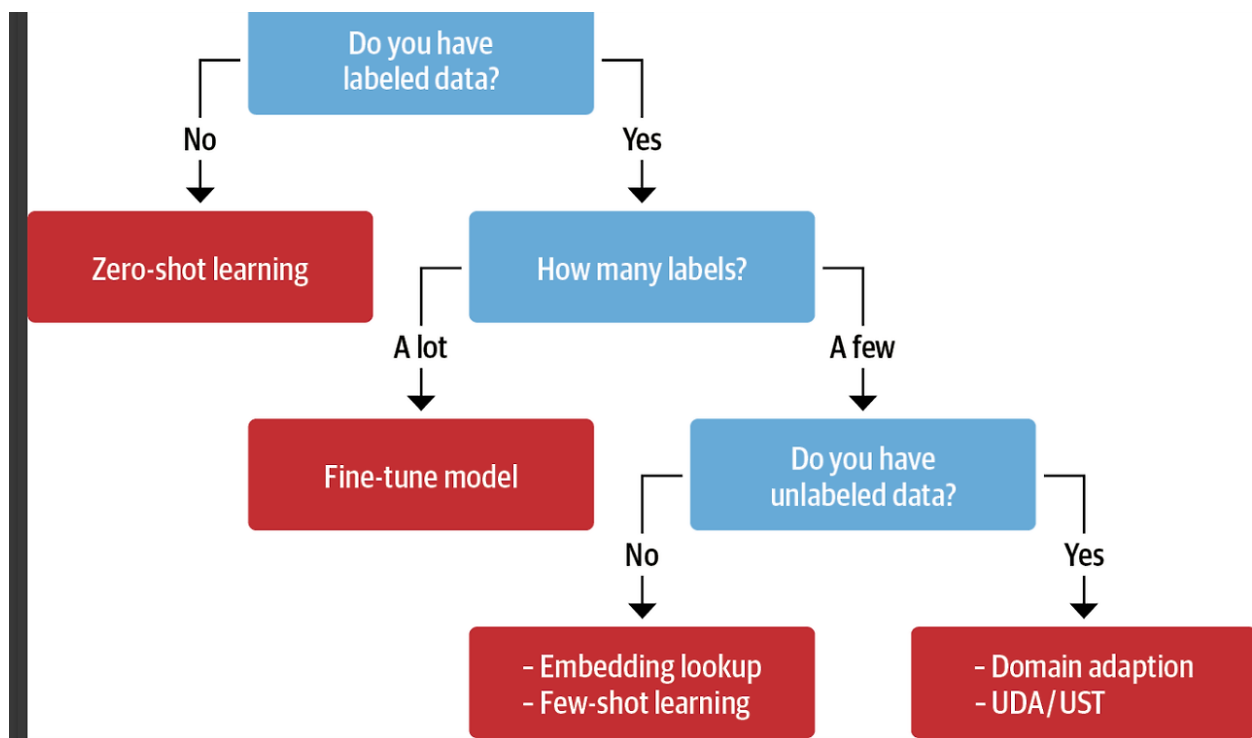
강아지 사진1-3 강아지

강아지 사진2 강아지

고양이 사진1 고양이

고양이 사진2 고양이

X(unlabeled data- 문장들) y*(labeled data-> x,y 쌍으로 있는 data)



라벨을 정하는 방법(모델의 예측값을 정하는 방법) -제로샷러닝에서(이미 학습한 모델 불러오는 것)

topk 방법

threshold(특정 확률 이상 값)

데이터 증식 (augmentation)

비슷한 의미를 가지면서 다른 데이터로 변형해줘야 함 (동의어 대체, 랜덤으로 단어를 바꾸거나 지우거나)

$x \quad y$

$x' \quad y$

$x'' \quad y$

ex) synonym 활용해서 증식

ex) backtranslation -중간 다른언어를 거쳐서 다시 번역해서 다른 데이터 만들기

한국어 -> 영어 -> 한국어

```

from transformers import set_seed
import nlpaug.augmenter.word as naw
import nlpaug.augmenter.char as nac
import nlpaug.augmenter.sentence as nas
  
```

```

import nlpaug.flow as nafc
import nltk
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('omw-1.4')

set_seed(3)
text = "Even if you defeat me Megatron, others will rise to defeat your tyranny"
aug = {}
aug["synonym_replace"] = naw.SynonymAug(aug_src='wordnet')
aug["random_insert"] =
naw.ContextualWordEmbsAug(model_path="distilbert-base-uncased",
                           device="cpu", action="insert",
                           aug_max=1)
aug["random_swap"] = naw.RandomWordAug(action="swap")
aug["random_delete"] = naw.RandomWordAug()
aug["bt_en_de"] = naw.BackTranslationAug(
    from_model_name='facebook/wmt19-en-de',
    to_model_name='facebook/wmt19-de-en'
)
for k,v in aug.items():
    print(f"원본 텍스트: {text}")
    print(f"{k}: {v.augment(text)}")
    print("")

```

contextual word embedding

문맥을 고려한 word embedding

vector의 유사도를 계산해서 유의어를 찾을수 있다.

```

from transformers import set_seed
import nlpaug.augmenter.word as naw

set_seed(3)
aug = naw.ContextualWordEmbsAug(model_path="distilbert-base-

```

```
uncased",

                                device="cpu", action="substitute")

text = "Transformers are the most popular toys"
print(f"원본 텍스트: {text}")
print(f"증식된 텍스트: {aug.augment(text)}")
```

원본 텍스트: Transformers are the most popular toys
증식된 텍스트: ['transformers – the most coveted toys']

```
def augment_text(batch, transformations_per_example=1):
    text_aug, label_ids = [], []
    for text, labels in zip(batch["text"], batch["label_ids"]):
        text_aug += [text]
        label_ids += [labels]
        for _ in range(transformations_per_example):
            text_aug += aug.augment(text) #데이터를 증식해서
            label_ids += [labels]
    return {"text": text_aug, "label_ids": label_ids}
```

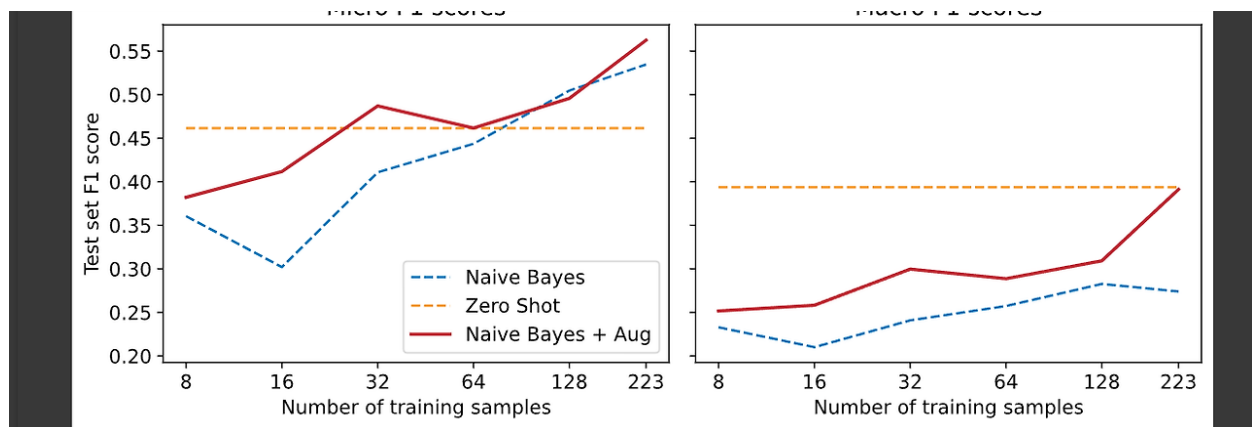
```
for train_slice in train_slices: #naive basic 모델에서 학습한다.
    # 훈련 슬라이스와 테스트 데이터를 준비합니다
    ds_train_sample = ds["train"].select(train_slice)
    # 증식 결과를 펼쳐서 레이블에 정렬합니다!
    ds_train_aug = (ds_train_sample.map(
        augment_text, batched=True,
    remove_columns=ds_train_sample.column_names)
        .shuffle(seed=42))
    y_train = np.array(ds_train_aug["label_ids"])
    y_test = np.array(ds["test"]["label_ids"])
    # 간단한 CountVectorizer를 사용해 텍스트를 토큰 카운트로 인코딩합니다
    count_vect = CountVectorizer()
    X_train_counts = count_vect.fit_transform(ds_train_aug["text"])
    X_test_counts = count_vect.transform(ds["test"]["text"])
    # 모델을 만들고 훈련합니다!
    classifier = BinaryRelevance(classifier=MultinomialNB())
```

```

classifier.fit(X_train_counts, y_train)
# 예측을 생성하고 평가합니다
y_pred_test = classifier.predict(X_test_counts)
clf_report = classification_report(
    y_test, y_pred_test, target_names=mlb.classes_,
    zero_division=0,
    output_dict=True)
# 평가 결과를 저장합니다
macro_scores["Naive Bayes + Aug"].append(clf_report["macro avg"]
["f1-score"])
micro_scores["Naive Bayes + Aug"].append(clf_report["micro avg"]
["f1-score"])

```

데이터 증식하면 성능이 좋아진걸 볼수있다.



벡터의 위치를 2D로 보여준것

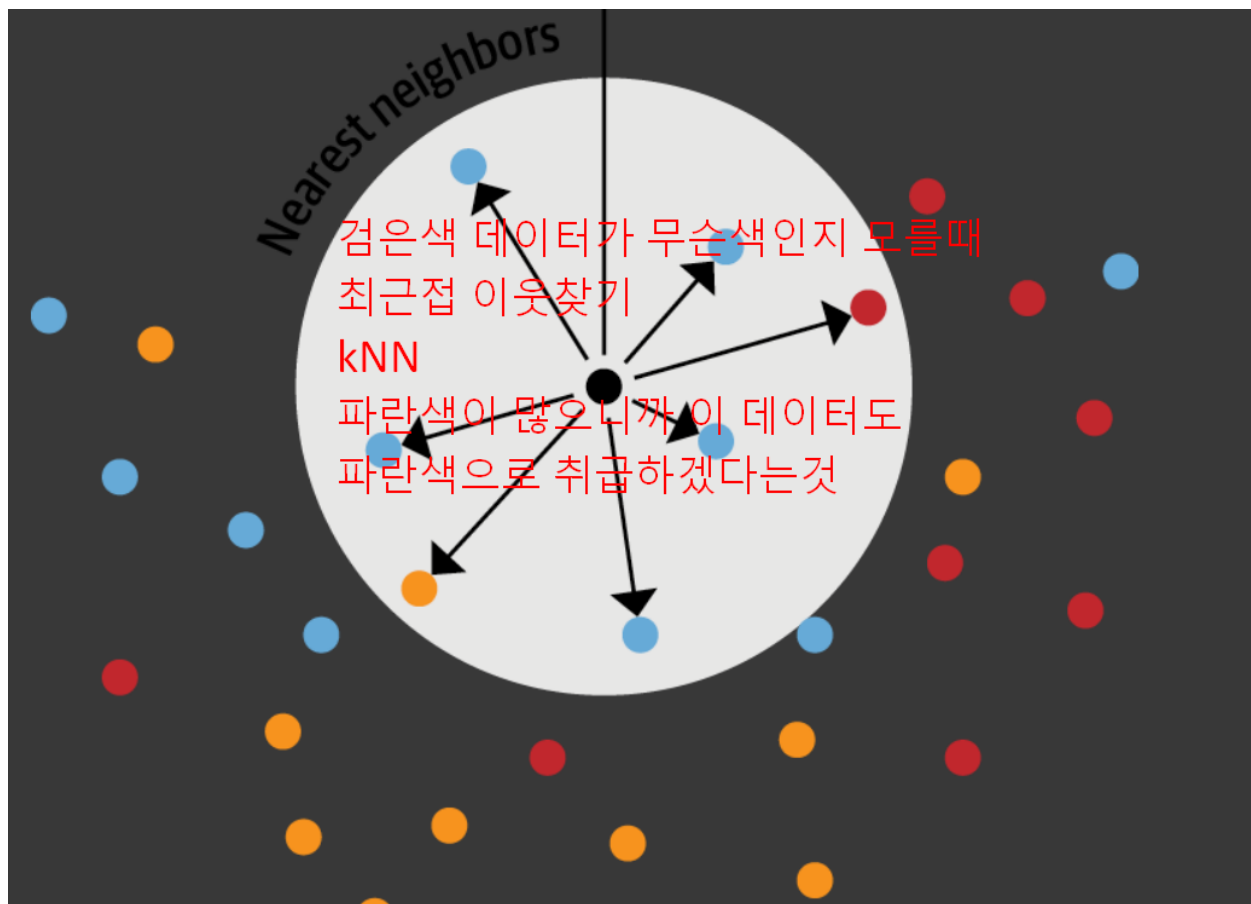
검은색 데이터가 무슨색인지 모를때

최근접 이웃찾기

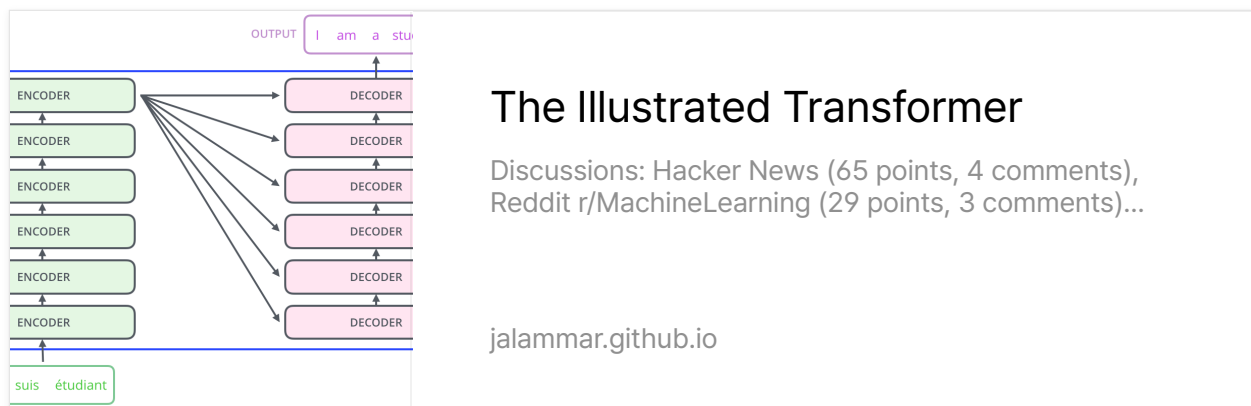
kNN

파란색이 많으니까 이 데이터도

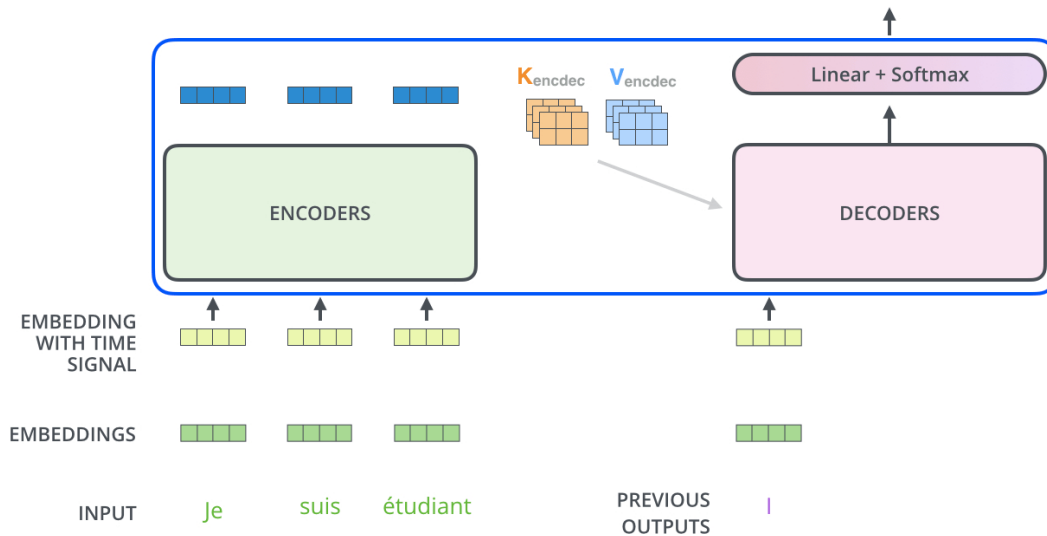
파란색으로 취급하겠다는것



출처: <https://jalammar.github.io/illustrated-transformer/>



encoding에서 embedding 으로 나온 벡터를 다 더해서 문장을 표현하는 벡터로 취급하자



아래그림) 쿼리 텍스트를 입력해서 가까운 위치의 세개 문장을 갖고올때
label이 동일하게 new model 이 되었음

```
i, k = 0, 3      # 첫 번째 쿼리와 3개의 최근접 이웃을 선택합니다
rn, nl = "\r\n\r\n", "\n"  # 간결한 출력을 위해 텍스트에서 줄바꿈 문자를 삭제합니다

query = np.array(embs_valid[i]["embedding"], dtype=np.float32)
scores, samples = embs_train.get_nearest_examples("embedding",
query, k=k)

print(f"쿼리 레이블: {embs_valid[i]['labels']}")
print(f"쿼리 텍스트:\n{embs_valid[i]['text'][:200].replace(rn, nl)}
[...]\n")
print("="*50)
print(f"추출된 문서:")
for score, label, text in zip(scores, samples["labels"],
samples["text"]):
    print("="*50)
    print(f"텍스트:\n{text[:200].replace(rn, nl)} [...]")
    print(f"점수: {score:.2f}")
    print(f"레이블: {label}")
```

아래그림) 쿼리 텍스트를 입력해서 가까운 위치의 세개 문장을 갖고올때

label이 동일하게 new model 이 되었음
최근 인접한 세개 문장을 갖고온것

```
# 🌟 New model addition
## Model description
### Linformer: Self-Attention with Linear Complexity
Paper published June 9th on ArXiv: https://arxiv.org/abs/2006.04768
La [...]
점수: 54.92
레이블: ['new model']

=====

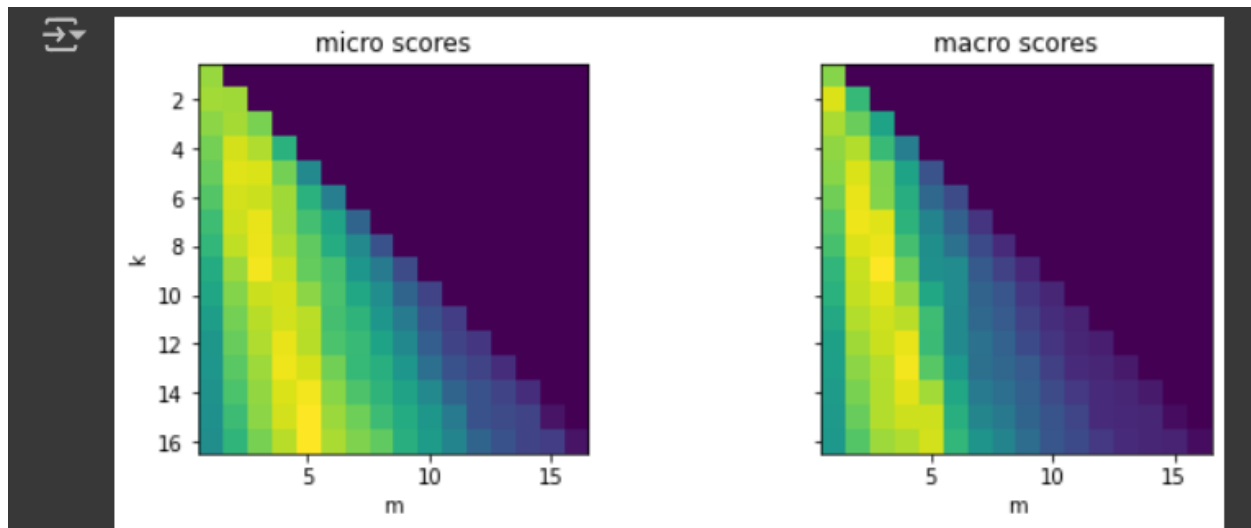
텍스트:
Add FAVOR+ / Performer attention

# 🌟 FAVOR+ / Performer attention addition
Are there any plans to add this new attention approximation block to
Transformers library?
## Model description
The n [...]
점수: 57.90
레이블: ['new model']

=====

텍스트:
Implement DeLight: Very Deep and Light-weight Transformers

# 🌟 New model addition
## Model description
DeLight, that delivers similar or better performance than
transformer-based models with sign [...]
점수: 60.12
레이블: ['new model']
```



```
k, m = np.unravel_index(perf_micro.argmax(), perf_micro.shape) # k는
근처에서 몇개 데이터 볼지, m은 라벨을 할당하기 위한 threshold임계값
print(f"최상의 k: {k}, 최상의 m: {m}")
```

k=5, m=3 (threshold) knn이 근처에서 몇개살펴볼지, m값은 경계값(임계값)

근처의 5개

그 5개의 라벨 살펴보니까

[A, B]

[A, C]

[B]

[A]

[A, B, C, D]

Query의 라벨은 뭐라고 예측되어야 하는걸까? [A,B] -> m은 나온갯수가 3번 나와야

라벨은 [A,B]

Content

System (prompt) -전 제조건	You are an AI assistant with a passion for creative writing and storytelling. Your task is to collaborate with users to create engaging stories, offering imaginative plot twists and dynamic character development. Encourage the user to contribute their ideas and build upon them to create a captivating narrative.
User(입력- 구체적작업 알 려줌)	Let's create a story about a young woman named Lila who discovers she has the power to control the weather. She lives in a small town where everyone knows each other.

입력 - 위 다 넣어서

input - 일반적 연구활동시 위 내용 다 넣을때 쓰이는것

프롬프트를 사용한 인-컨텍스트 학습과 퓨-샷 학습 - 모델이 커야 가능

```
prompt = """\n
Translate English to Korean:\n
thanks => 고마워\n
sorry => 미안해\n
hello => 안녕 # 여기까지가 prompt\n
i go to school =>\n
"""
```

라벨 달린상태

x1, y1

x2, y2

...

x200, y200 -> naive bayesian model - baseline model

라벨 안달린상태

x201

x202

x203

....

x9200

pretrained model - 수많은 데이터에 대해 사전에 학습된 모델 (아직 특정한 task는 실행전-결과를 모름)

- **zero shot learning** (이 task대해 아무학습없이 task수행해봄)

(BERT, GPT, ... 인터넷에서 긁어온 엄청나게 많은 텍스트,

pretraining task - MLM-masked language model -가운데 단어맞추기

pretraining task - Causal Language Modeling - 다음 단어 맞추기)

domain adaptation - 학습(pretrain아님)

pretrain 할때 학습하던 task를 unlabel data에도 수행해보기

ex.) 아이가 단어를 배우면서 언어를 이해한 상태

ex.) 200~9200 의료관련 라벨안달린 자료 주고 , 가운데 단어 맞추기 같은 pretrain task를 수행하면 의료에 대한 지식 늘어남.

라벨 달린 데이터 주고 드디어 실행할 task 학습을 수행

Pretrain

1. pretrain 할때 학습하던 task를 unlabel data에도 수행해보기

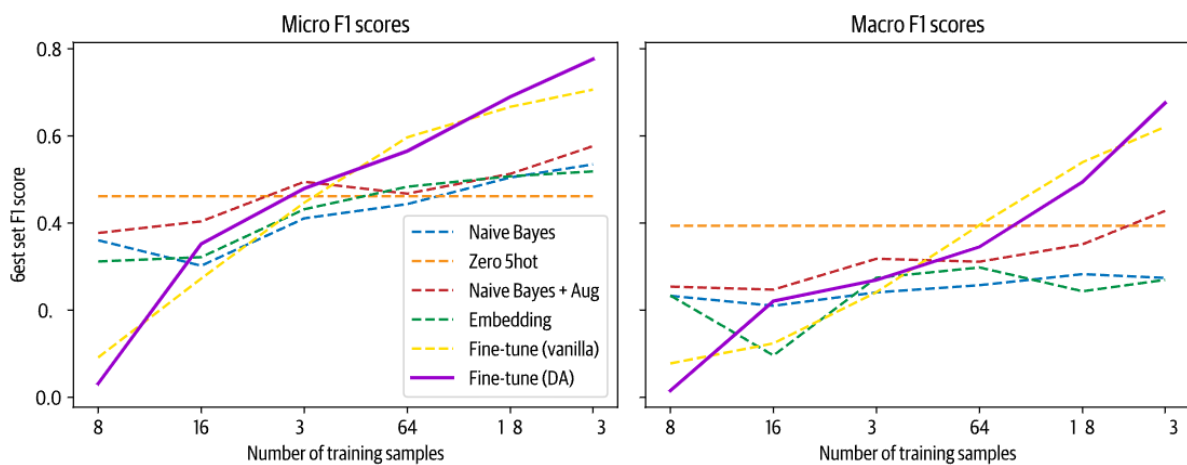
Domain Adaptation

2. ex.) 200~9200 의료관련 라벨안달린 자료 주고 , 가운데 단어 맞추기 같은 pretrain task를 수행하면 의료에 대한 지식 늘어남.

Fine-tuning

3. 라벨 달린 데이터 주고 드디어 실행할 task 학습을 수행

성과는 pretrain -> Domain Adaptation -> Fine-tuning 가 성능이 더 잘나옴



♡ 2



0000

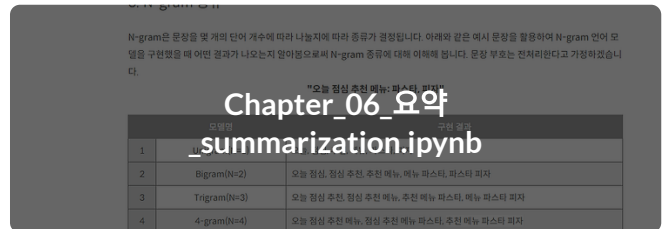
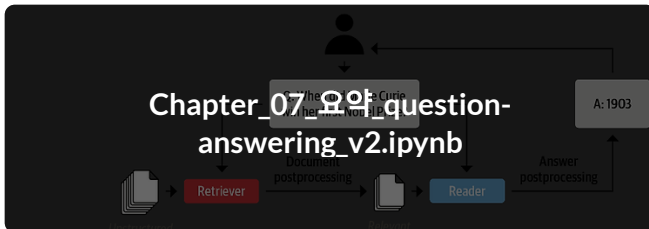
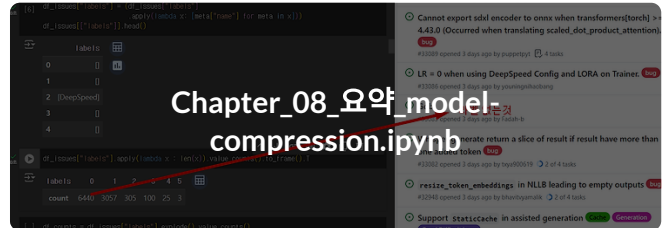
구독하기

'transformer' 카테고리의 다른 글

입학전형 (0)	2024.08.30
Chapter_08 요약_model-compression.ipynb (0)	2024.08.26
Chapter_07 요약_question-answering_v2.ipynb (0)	2024.08.16
Chapter_06 요약_summarization.ipynb (0)	2024.08.07

관련글

관련글 더보기



자연어(NLP)

네이쳐2024 님의 블로그입니다.

구독하기 +

댓글 0



이름

비밀번호

내용을 입력하세요.



등록