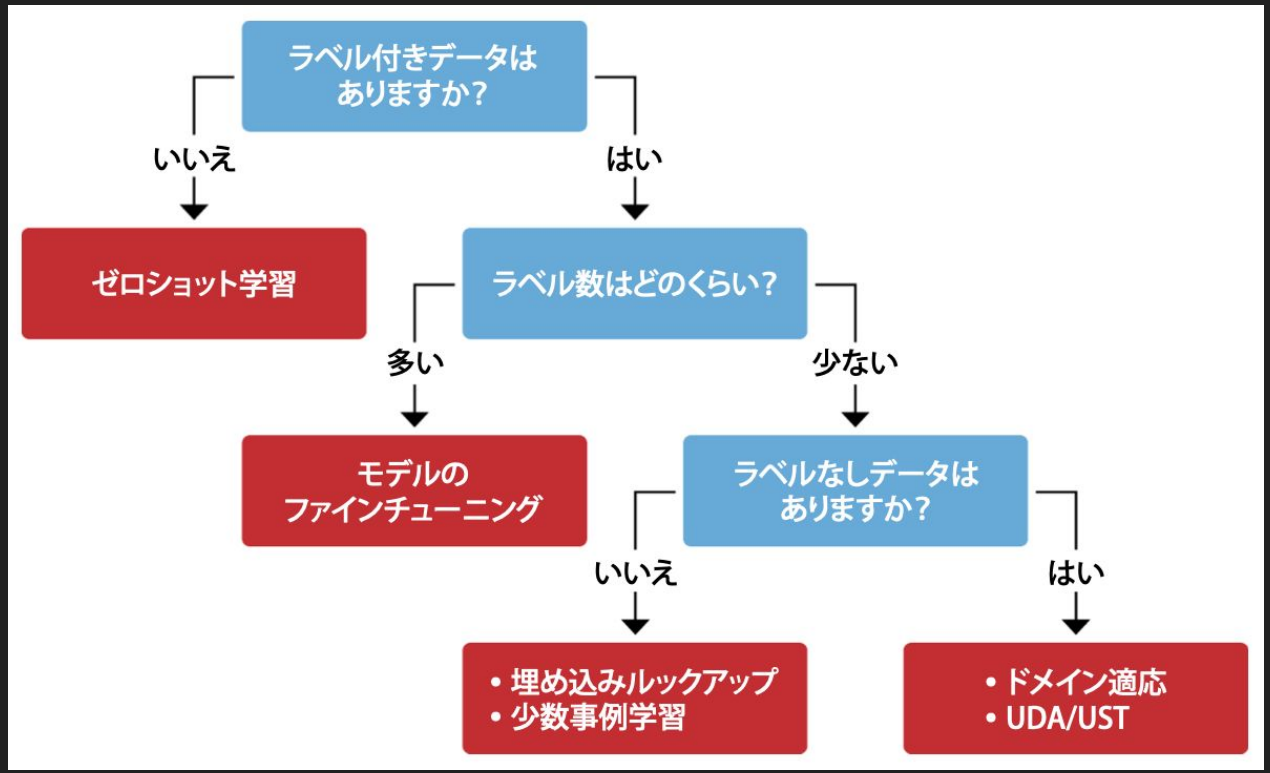
Natural Language Processing with Transformers
9章 ラベル不足への対応方法

2023/08/13

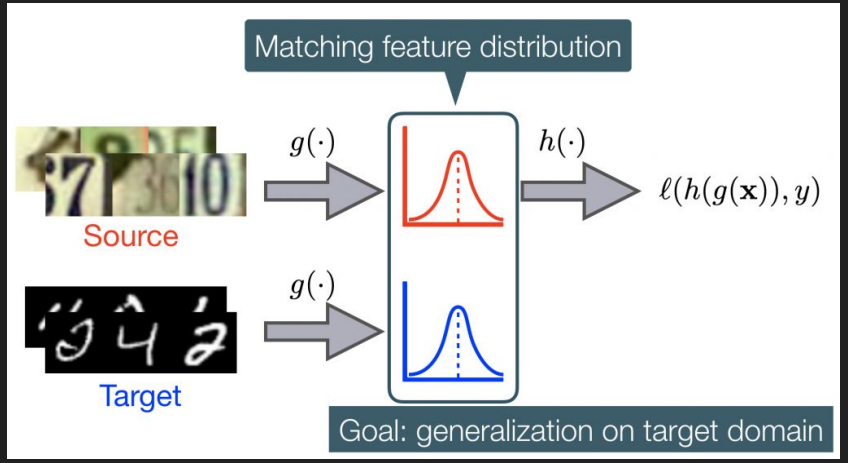
趙 将司（川口 将司）

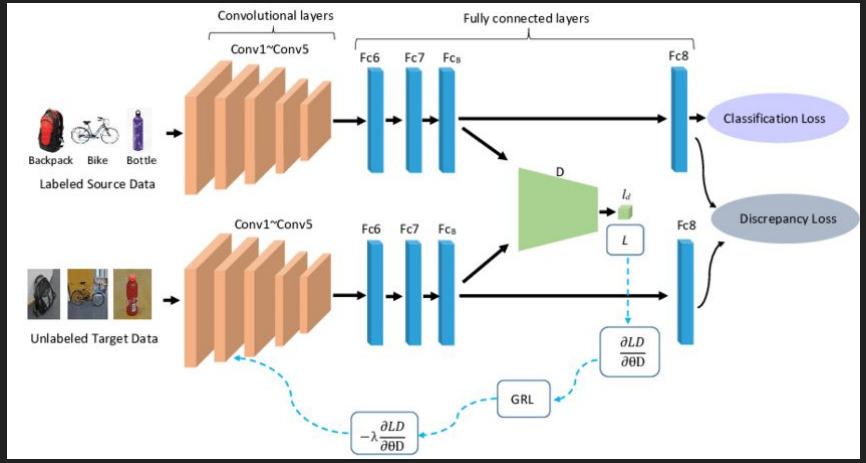
ラベル有無の割合に応じた最適手法の選定



ドメイン適応

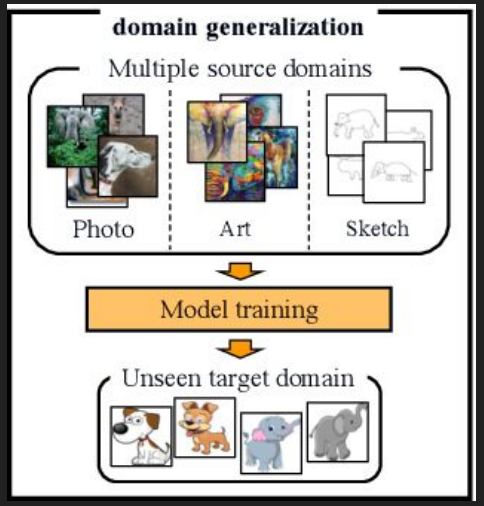
転移学習の一種。
ラベルが豊富なソースドメインでの学習成果を、ラベルが少量のターゲットドメインでの類似問題に適用し、高い性能を実現する手法。

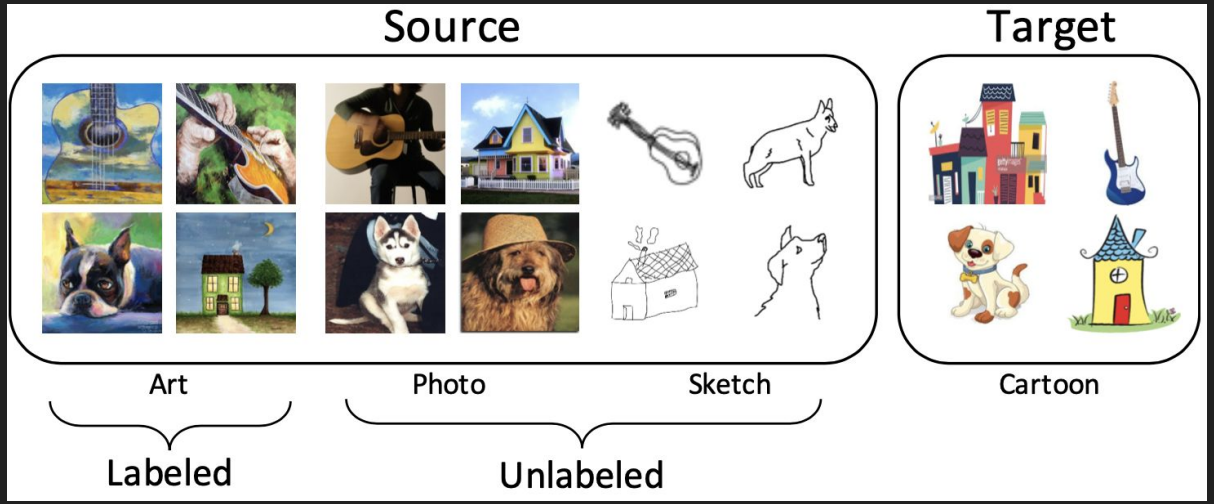




ドメイン汎化

ターゲットドメインのデータが一切与えられない前提、つまり未知のデータ（unseen data）に対処できるモデルを目指す手法。





BERT（MLM）によるテキスト分類問題

BERTのベースモデルでも簡単なテキスト分類に対処可能。

```
from transformers import pipeline

pipe = pipeline("fill-mask", model="bert-base-uncased")
```

```
movie_desc = "The main characters of the movie madacascar \
are a lion, a zebra, a giraffe, and a hippo. "
prompt = "The movie is about [MASK]."
```

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

```
Token animals:   0.103%
Token cars:      0.001%
```

```
movie_desc = "In the movie transformers aliens \
can morph into a wide range of vehicles."
```

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

```
Token cars:      0.139%
Token animals:   0.006%
```

自然言語推論（natural language inference:NLI）

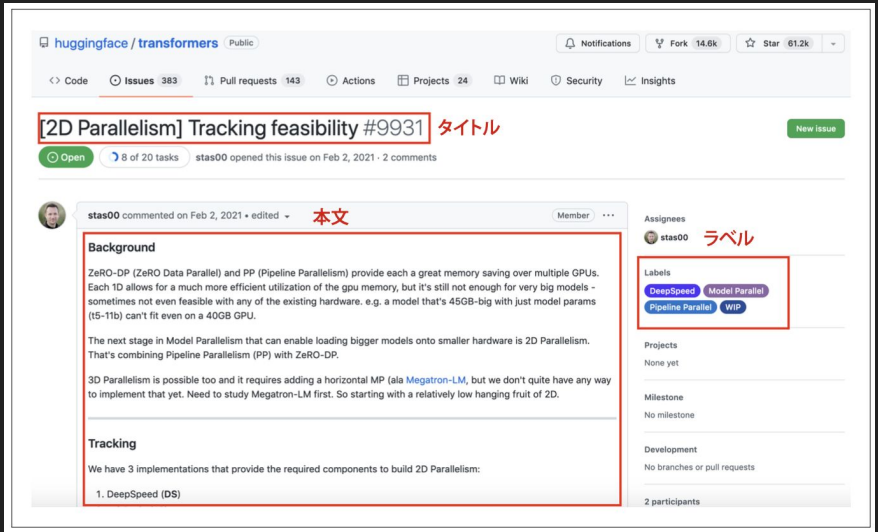
含意関係認識のタスクでは、前提（Premise）のテキストに対して仮説（Hypothesis）のテキストが含意するか（entailment）、矛盾するか（contradiction）、中性的（neutral）か、いずれかのラベルを推論する。

主なデータセットとして「Multi-Genre NLI コーパス(MNLI)」「Cross-Lingual NLI コーパス(XNLI)」が挙げられる。

Premise	Label	Hypothesis
A man inspects the uniform of a figure in some East Asian country.	contradiction	The man is sleeping.
An older and younger man smiling.	neutral	Two men are smiling and laughing at the cats playing on the floor.
A soccer game with multiple males playing.	entailment	Some men are playing a sport.

Githubのissue内容に対するラベルの推論

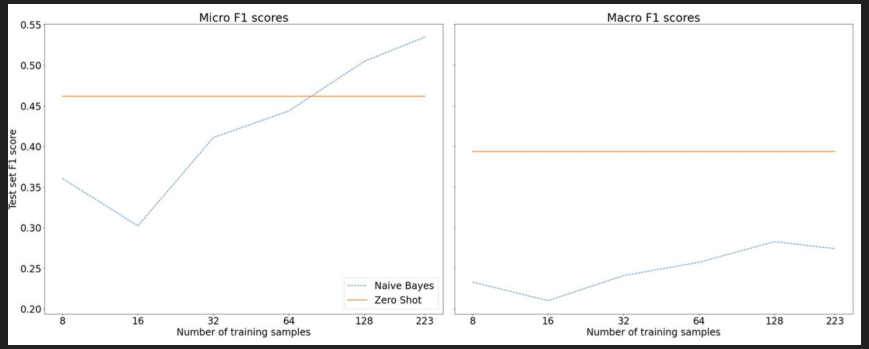
Githubのissueの本文に照らして、適切なラベルを付与するタスクがテーマ。
ナイーブベイズをベースラインとする。




```
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']
```

MNLIに基づくゼロショット分類モデルとナイーブベイズの比較

- ラベル付きデータが50件～60件程度の場合、ゼロショット分類モデル zero-shot-classification（top-1）の性能はナイーブベイズを上回る。
- また50件以上の学習を経ても、マクロF1指標でゼロショットパイプラインの性能が圧倒的に優れる（ナイーブベイズは少数クラスの学習が困難）。



各Issueに付与されるラベルの数（大部分はラベル無し）

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

もっとも頻繁に使われる上位 8 件のラベル

Number of labels: 65

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

自然言語処理におけるデータ拡張

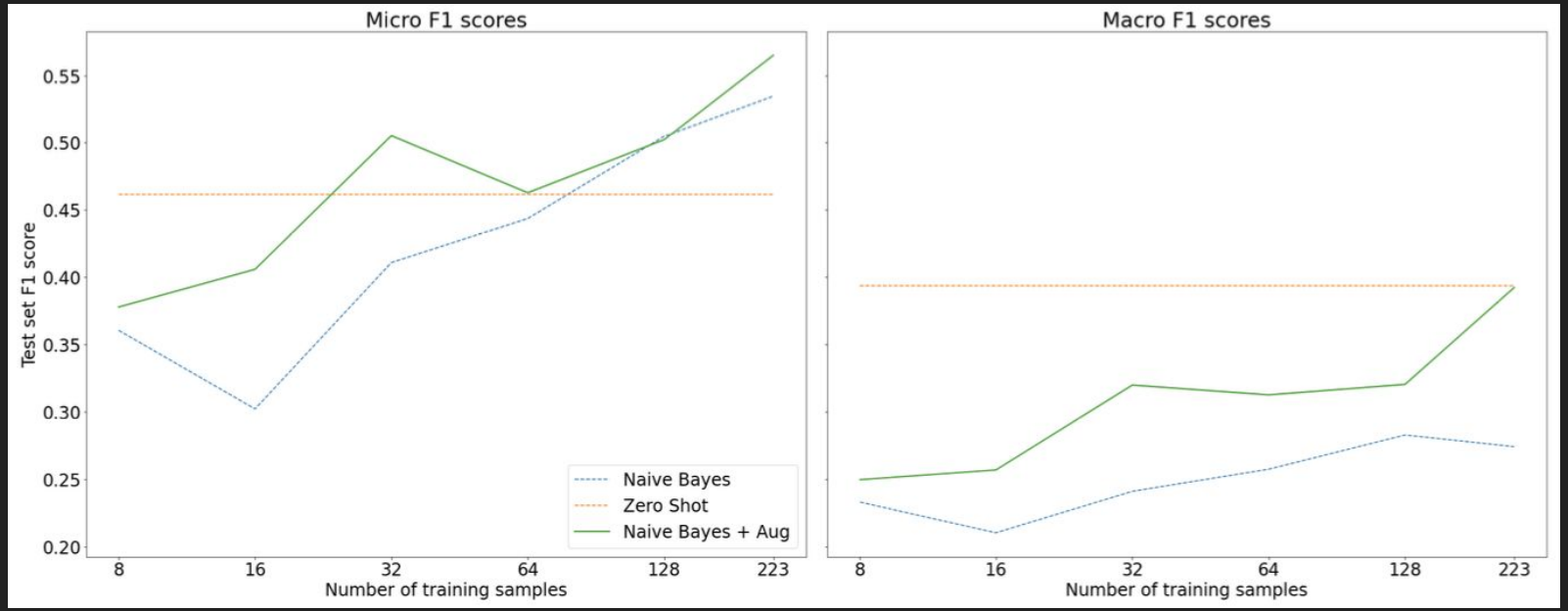
アフィン変換等で容易にデータを水増し可能な画像データと異なり、テキストデータは語彙の構成や並び順に対してラベルがセンシティブな傾向がある。
自然言語処理において主なデータ拡張のアプローチとしては、以下の2種類が挙げられる。

- バックトランスレーション
  - 機械翻訳等でA語->B語に翻訳した文章を、改めてB語->A語に翻訳しなおす。
  - 一般的に機械翻訳は一般語を採用する傾向があるため、専門用語が必要とされるコーパス構築には不向きともいえる（一般語でコーパスを構築したい場合は、むしろ向いている）。
- トークン摂動
  - 同義語置換、単語挿入、入れ替え、削除などの簡単な変換をランダムに加える。

表9-2　テキストデータに対するさまざまなデータ拡張の技術

データ拡張の技術	文
何もしない場合	Even if you defeat me Megatron, others will rise to defeat your tyranny
同義語置換	Even if you kill me Megatron, others will prove to defeat your tyranny
ランダムな挿入	Even if you defeat me Megatron, others humanity will rise to defeat your tyranny
ランダムな入れ替え	You even if defeat me Megatron, others will rise defeat to tyranny your
ランダムな削除	Even if you me Megatron, others to defeat tyranny
バックトランスレーション（ドイツ語）	Even if you defeat me, others will rise up to defeat your tyranny

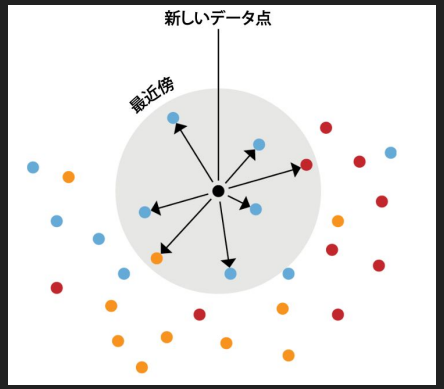
データ拡張後の学習に基づくナイーブベイズとの比較



埋め込みルックアップテーブルを活用したテキスト分類

以下の要領でテキスト分類を試みる。

1. まず事前にラベルが付与されたテキストデータを、所定の言語モデルに基づいた埋め込み表現（ベクトル）に変換・格納しておく（※）。
2. 推論（分類）時、入力テキストも埋め込み表現へ変換し、 1.で変換・格納されたベクトルと比較した最近傍探索（下図）を行う。
3. 周辺の埋め込みに対応するラベルを集計し、入力テキストに対するラベル付けを行う。



ドットの位置：埋め込み表現の座標
ドットの色　：ラベル
中心の黒点　：入力テキストの埋め込み表現

※一般的な言語モデルは1トークンに対して1つの埋め込みベクトルを返す。複数のトークンで構成されるテキストは、全トークンの埋め込み表現を平均する等のプーリング処理で1つの埋め込みに圧縮する。ただし、平均値にバイアスがかからないようpaddingやmask等の特殊なトークンは除外する。

（参考）埋め込みルックアップを構築～最近傍探索のコード

```python
import torch
from transformers import AutoTokenizer, AutoModel

model_ckpt = "miguelvictor/python-gpt2-large"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
model = AutoModel.from_pretrained(model_ckpt)

def mean_pooling(model_output, attention_mask):
    # トークン埋め込みの抽出
    token_embeddings = model_output[0]
    # アテンションマスクの計算
    input_mask_expanded = (attention_mask
                           .unsqueeze(-1)
                           .expand(token_embeddings.size())
                           .float())
    # マスクトークンを無視して、埋め込みの和を取る
    sum_embeddings = torch.sum(token_embeddings * input_mask_expanded, 1)
    sum_mask = torch.clamp(input_mask_expanded.sum(1), min=1e-9)
    # 平均ベクトルを返す
    return sum_embeddings / sum_mask

def embed_text(examples):
    inputs = tokenizer(examples["text"], padding=True, truncation=True,
                       max_length=128, return_tensors="pt")
    with torch.no_grad():
        model_output = model(**inputs)
    pooled_embeds = mean_pooling(model_output, inputs["attention_mask"])
    return {"embedding": pooled_embeds.cpu().numpy()}
```

ラベル付きデータの埋め込み

```python
tokenizer.pad_token = tokenizer.eos_token
embs_train = ds["train"].map(embed_text, batched=True, batch_size=16)
embs_valid = ds["valid"].map(embed_text, batched=True, batch_size=16)
embs_test = ds["test"].map(embed_text, batched=True, batch_size=16)
```

Faissへの格納

```python
embs_train.add_faiss_index("embedding")
```

入力テキストの埋め込みと最近傍探索

```python
i, k = 0, 3 # 最初のクエリと3つの最近傍を選択
rn, nl = "\r\n\r\n", "\n" # テキスト中の改行を削除するために使用

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

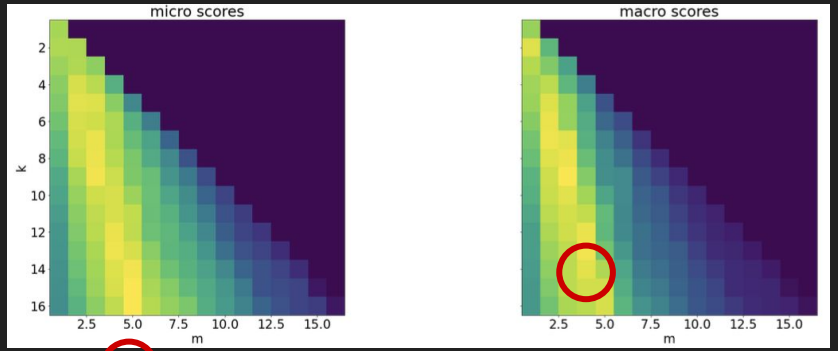
```python
for score, label, text in zip(scores, samples["labels"], samples["text"]):
    print("="*50)
    print(f"TEXT:\n{text[:200].replace(rn, nl)} [...]")
    print(f"SCORE: {score:.2f}")
    print(f"LABELS: {label}")
```

ラベルの採用可否を決定する閾値の最適化

「最近傍探索の上位 {k} サンプルに付与された各ラベルの出現数を集計。当該ラベルの出現数が閾値 {m} を超えたとき、当該ラベルを採用する」という実装において、最適なkとmを探索したい。

```python
def get_sample_preds(sample, m):
    return (np.sum(sample["label_ids"], axis=0) >= m).astype(int)

def find_best_k_m(ds_train, valid_queries, valid_labels, max_k=17):
    max_k = min(len(ds_train), max_k)
    perf_micro = np.zeros((max_k, max_k))
    perf_macro = np.zeros((max_k, max_k))
    for k in range(1, max_k):
        for m in range(1, k + 1):
            _, samples = ds_train.get_nearest_examples_batch("embedding",
                                                             valid_queries, k=k)
            y_pred = np.array([get_sample_preds(s, m) for s in samples])
            clf_report = classification_report(valid_labels, y_pred,
                target_names=mlb.classes_, zero_division=0, output_dict=True)
            perf_micro[k, m] = clf_report["micro avg"]["f1-score"]
            perf_macro[k, m] = clf_report["macro avg"]["f1-score"]
    return perf_micro, perf_macro
```

k, mの設置値毎のF1スコア




```python
k, m = np.unravel_index(perf_micro.argmax(), perf_micro.shape)
print(f"Best k: {k}, best m: {m}")
```

```
Best k: 15, best m: 5
```

比較的シンプルなファインチューニング

```
import torch
from transformers import (AutoTokenizer, AutoConfig,
                          AutoModelForSequenceClassification)

model_ckpt = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)

def tokenize(batch):
    return tokenizer(batch["text"], truncation=True, max_length=128)
ds_enc = ds.map(tokenize, batched=True)
ds_enc = ds_enc.remove_columns(['labels', 'text'])
```

```
ds_enc.set_format("torch")
ds_enc = ds_enc.map(lambda x: {"label_ids_f": x["label_ids"].to(torch.float)},
                    remove_columns=["label_ids"])
ds_enc = ds_enc.rename_column("label_ids_f", "label_ids")
```

```
from transformers import Trainer, TrainingArguments

training_args_fine_tune = TrainingArguments(
    output_dir="./results", num_train_epochs=20, learning_rate=3e-5,
    lr_scheduler_type='constant', per_device_train_batch_size=4,
    per_device_eval_batch_size=32, weight_decay=0.0,
    evaluation_strategy="epoch", save_strategy="epoch",logging_strategy="epoch",
    load_best_model_at_end=True, metric_for_best_model='micro f1',
    save_total_limit=1, log_level='error')
```

```
from scipy.special import expit as sigmoid

def compute_metrics(pred):
    y_true = pred.label_ids
    y_pred = sigmoid(pred.predictions)
    y_pred = (y_pred>0.5).astype(float)

    clf_dict = classification_report(y_true, y_pred, target_names=all_labels,
                                     zero_division=0, output_dict=True)
    return {"micro f1": clf_dict["micro avg"]["f1-score"],
            "macro f1": clf_dict["macro avg"]["f1-score"]}
```

```
config = AutoConfig.from_pretrained(model_ckpt)
config.num_labels = len(all_labels)
config.problem_type = "multi_label_classification"

for train_slice in train_slices:
    model = AutoModelForSequenceClassification.from_pretrained(model_ckpt,
                                                               config=config)
    trainer = Trainer(
        model=model, tokenizer=tokenizer,
        args=training_args_fine_tune,
        compute_metrics=compute_metrics,
        train_dataset=ds_enc["train"].select(train_slice),
        eval_dataset=ds_enc["valid"],)

    trainer.train()
    pred = trainer.predict(ds_enc["test"])
    metrics = compute_metrics(pred)
    macro_scores["Fine-tune (vanilla)"].append(metrics["macro f1"])
    micro_scores["Fine-tune (vanilla)"].append(metrics["micro f1"])

plot_metrics(micro_scores, macro_scores, train_samples, "Fine-tune (vanilla)")
```

インコンテキスト学習と少数事例学習（Few-shot Learning）

モデルが大規模化する程、インコンテキスト学習は働きやすい。

```
prompt = """\
Translate English to French:
thanks =>
"""
```

プロンプトに対して望ましい回答の例を所定のプロセスで作り上げて、学習量を増幅する手法がある。

PET (Pattern Exploiting Training)

Pattern-Verbalizer Pair (PVP) によりサンプルをクローズ形式 (cloze-style) に変換する。以下2つのモジュールから構成される。

①パターン（Pattern）

<premise> ? <mask>, <hypothesis>

“Oil prices fall back” ? <mask>, “Oil prices rise”

②語彙の変換（Verbalizer）

“Not Entailment/Entailment" -> “No/Yes”

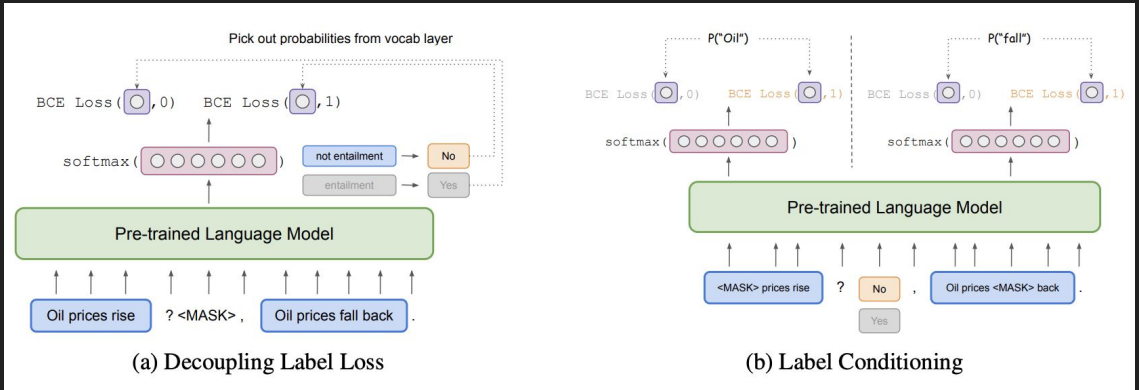
$$q(y|x) = \frac{\exp([\![G_m(x)]\!]_y)}{\sum_{y' \in \mathcal{Y}} \exp([\![G_m(x)]\!]_{y'})}$$

$$\mathcal{L} = \mathrm{CE}(q(y^*|x), y^*)$$

Tam et al. (2021)

ADAPET (A Densely-supervised Approach to Pattern Exploiting Training)

PETを拡張した手法。PVPに倣い負例を増幅し、学習している。



$$\mathcal{L}_D = \mathrm{BCE}(q(y^*|x), 1) + \sum_{y \neq y^*} \mathrm{BCE}(q(y|x), 0)$$
$$= \mathrm{CE}(q(y^*|x), y^*) - \sum_{y \neq y^*} \mathrm{CE}(q(y|x), y)$$

$$\mathcal{L}_D = \sum_{z^* \in y^*} \mathrm{BCE}(q(z^*|x), 1) + \sum_{y \neq y^*} \sum_{z \in y} \mathrm{BCE}(q(z|x), 0)$$

$$q(x^m|x', y) = \frac{\exp([\![G_m(x', y)]\!]_{x^m})}{\sum_{v' \in \mathcal{V}} \exp([\![G_m(x', y)]\!]_{v'})}$$

$$\mathcal{L}_M = \mathrm{BCE}(q(x^m|x', y^*), 1) + \sum_{y \neq y^*} \mathrm{BCE}(q(x^m|x', y), 0)$$

Tam et al. (2021)

ラベルなしデータによる言語モデルのファインチューニング

```python
def tokenize(batch):
    return tokenizer(batch["text"], truncation=True,
                     max_length=128, return_special_tokens_mask=True)

ds_mlm = ds.map(tokenize, batched=True)
ds_mlm = ds_mlm.remove_columns(["labels", "text", "label_ids"])
```

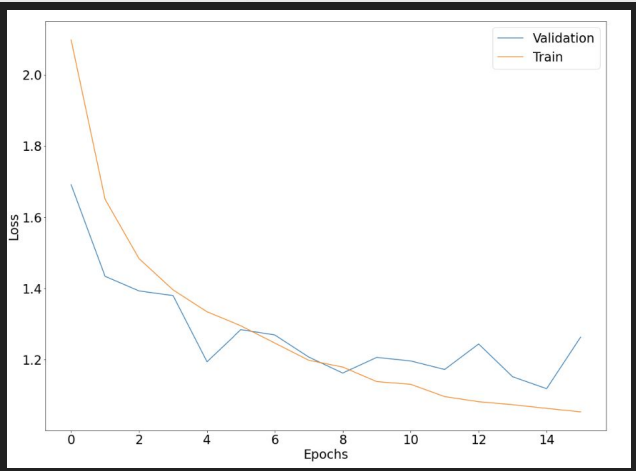
```python
from transformers import DataCollatorForLanguageModeling, set_seed

data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer,
                                                mlm_probability=0.15)
```

データコレーターにマスク付与を担わせる。

```python
set_seed(3)
data_collator.return_tensors = "np"
inputs = tokenizer("Transformers are awesome!", return_tensors="np")
outputs = data_collator([{"input_ids": inputs["input_ids"][0]}])

pd.DataFrame({
    "Original tokens": tokenizer.convert_ids_to_tokens(inputs["input_ids"][0]),
    "Masked tokens": tokenizer.convert_ids_to_tokens(outputs["input_ids"][0]),
    "Original input_ids": original_input_ids,
    "Masked input_ids": masked_input_ids,
    "Labels": outputs["labels"][0]}).T
```

	0	1	2	3	4	5
Original tokens	[CLS]	transformers	are	awesome	!	[SEP]
Masked tokens	[CLS]	transformers	are	awesome	[MASK]	[SEP]
Original input_ids	101	19081	2024	12476	999	102
Masked input_ids	101	19081	2024	12476	103	102
Labels	-100	-100	-100	-100	999	-100

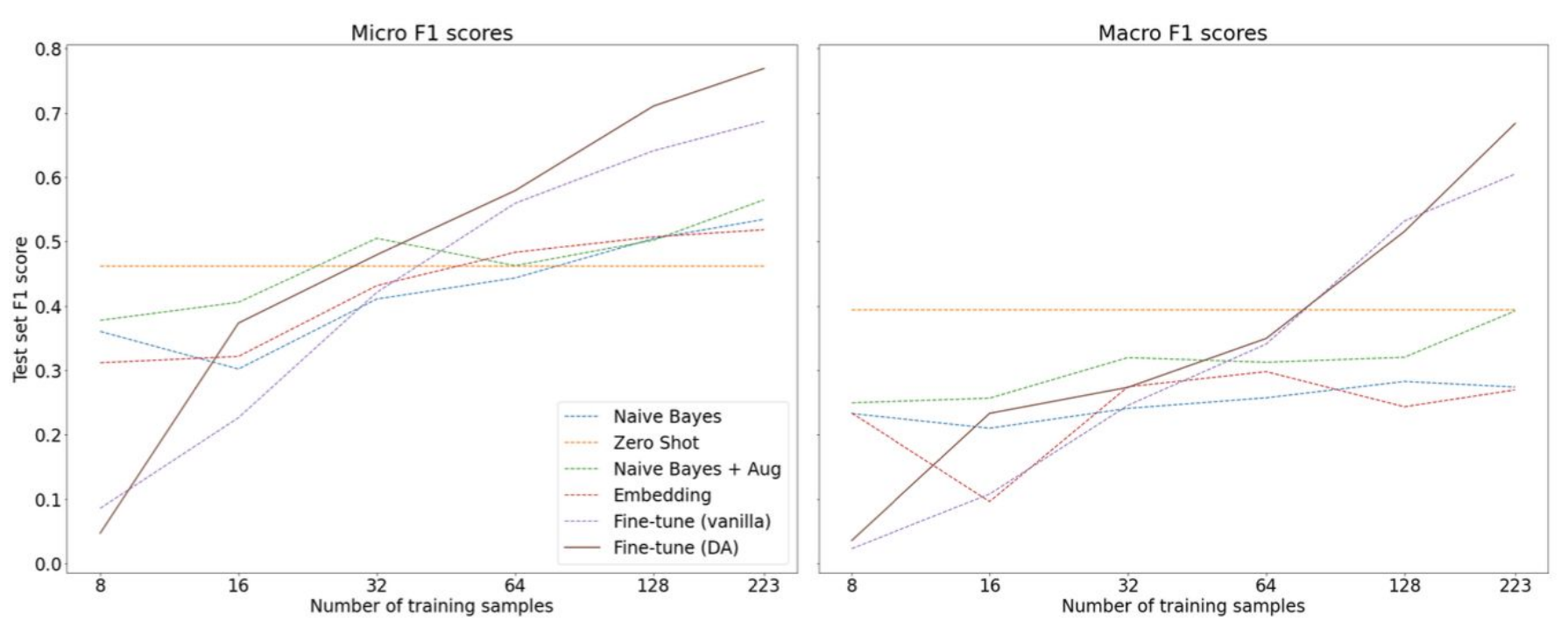
```python
from transformers import AutoModelForMaskedLM

training_args = TrainingArguments(
    output_dir = f"{model_ckpt}-issues-128", per_device_train_batch_size=32,
    logging_strategy="epoch", evaluation_strategy="epoch", save_strategy="no",
    num_train_epochs=16, push_to_hub=True, log_level="error", report_to="none")

trainer = Trainer(
        model=AutoModelForMaskedLM.from_pretrained("bert-base-uncased"),
        tokenizer=tokenizer, args=training_args, data_collator=data_collator,
        train_dataset=ds_mlm["unsup"], eval_dataset=ds_mlm["train"])

trainer.train()
trainer.push_to_hub("Training complete!")
```



ラベルなしデータによる分類器のファインチューニング

```
model_ckpt = f'{model_ckpt}-issues-128'
config = AutoConfig.from_pretrained(model_ckpt)
config.num_labels = len(all_labels)
config.problem_type = "multi_label_classification"

for train_slice in train_slices:
    model = AutoModelForSequenceClassification.from_pretrained(model_ckpt,
                                                               config=config)
trainer = Trainer(
    model=model,
    tokenizer=tokenizer,
    args=training_args_fine_tune,
    compute_metrics=compute_metrics,
    train_dataset=ds_enc["train"].select(train_slice),
    eval_dataset=ds_enc["valid"],
)

trainer.train()
pred = trainer.predict(ds_enc['test'])
metrics = compute_metrics(pred)
# DAはドメイン適応（Domain Adaptation）の略
macro_scores['Fine-tune (DA)'].append(metrics['macro f1'])
micro_scores['Fine-tune (DA)'].append(metrics['micro f1'])
```

ここまで挙げた手法の性能比較



UDA (Unsupervised Data Augmentation)

ソースドメインとターゲットドメインのサンプルに対する予測のKLダイバージェンスを最小化。

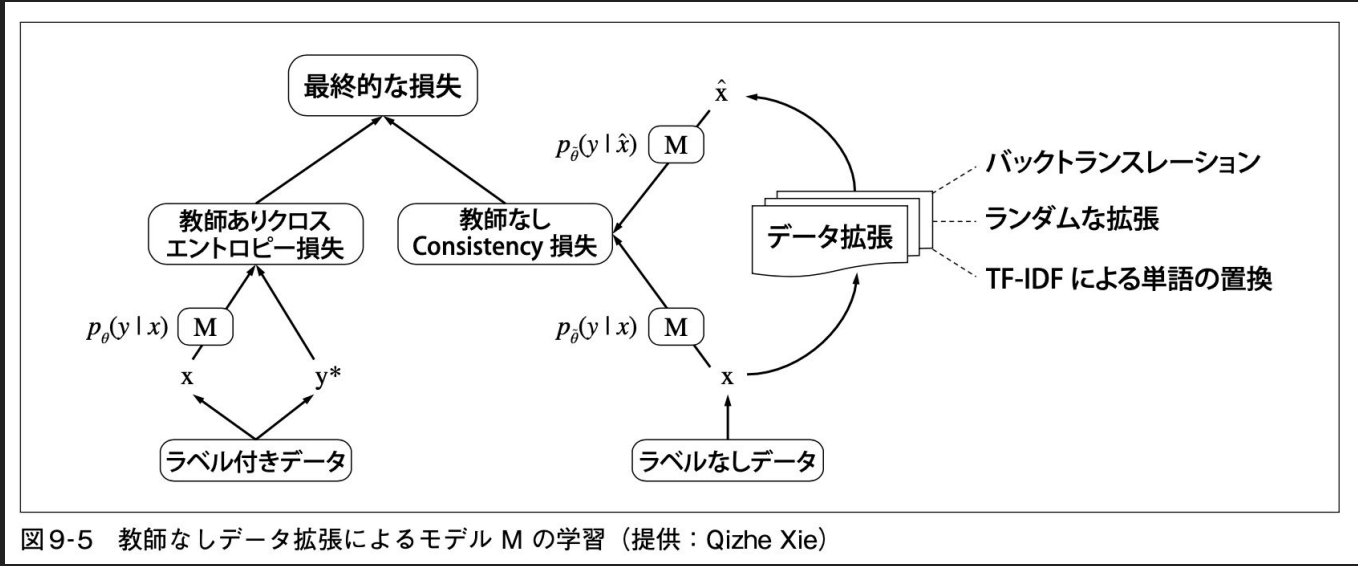


図9-5　教師なしデータ拡張によるモデル M の学習（提供：Qizhe Xie）

UST (Uncertainty-aware Self Training)

ラベル付きデータに基づくモデルを学習する（教師モデル）。
教師モデルを用いてラベル無しデータに対しラベル付けを行う（擬似ラベル）。
疑似ラベルの付いたデータで学習を行い、生徒モデルとする。これを次の教師モデルとする。

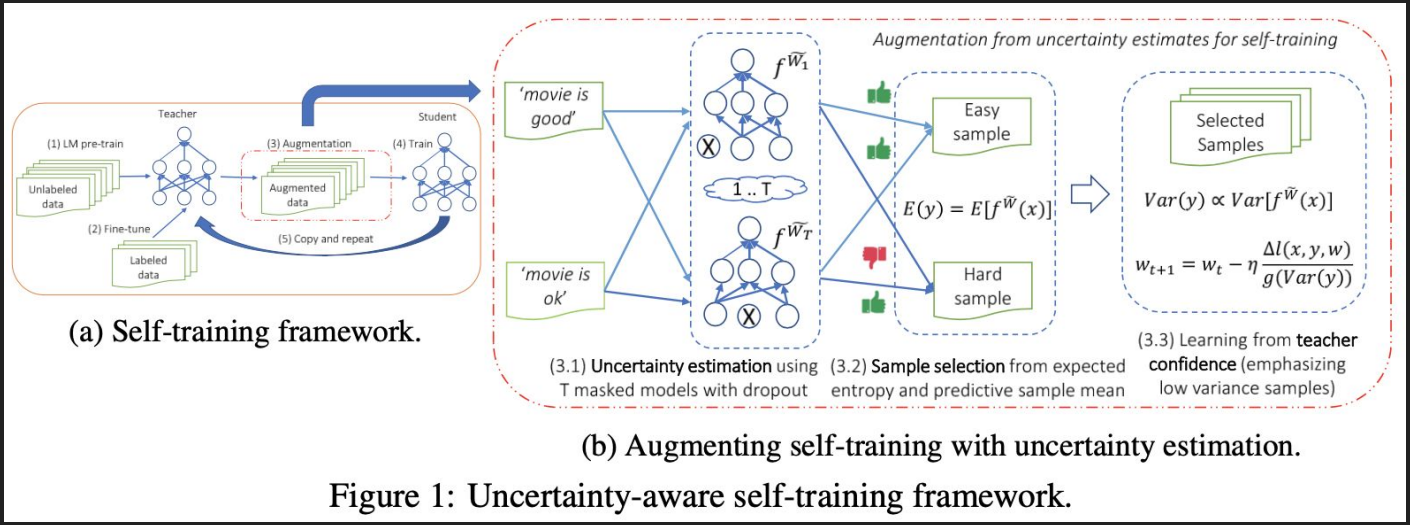


Figure 1: Uncertainty-aware self-training framework.