Text Classification on GLUE

Boom AI Presentation

Overview

The GLUE Benchmark is a group of nine classification tasks on sentences or pairs of sentences which are:

- CoLA (Corpus of Linguistic Acceptability) Determine if a sentence is grammatically correct or not.is a dataset containing sentences labeled grammatically correct or not.
- MNLI (Multi-Genre Natural Language Inference) Determine if a sentence entails, contradicts or is unrelated to a given hypothesis. (This dataset has two versions, one with the validation and test set coming from the same distribution, another called mismatched where the validation and test use out-of-domain data.)
- MRPC (Microsoft Research Paraphrase Corpus) Determine if two sentences are paraphrases from one another or not.
- QNLI (Question-answering Natural Language Inference) Determine if the answer to a question is in the second sentence or not. (This dataset is built from the SQuAD dataset.)
- QQP (Quora Question Pairs2) Determine if two questions are semantically equivalent or not.
- RTE (Recognizing Textual Entailment) Determine if a sentence entails a given hypothesis or not.
- SST-2 (Stanford Sentiment Treebank) Determine if the sentence has a positive or negative sentiment.
- STS-B (Semantic Textual Similarity Benchmark) Determine the similarity of two sentences with a score from 1 to 5.
- WNLI (Winograd Natural Language Inference) Determine if a sentence with an anonymous pronoun and a sentence with this pronoun replaced are entailed or not. (This dataset is built from the Winograd Schema Challenge dataset.)

Initialization

- Major packages to be utilized

```
# ! pip install datasets transformers

# from huggingface_hub import notebook_login
# notebook_login()

# !apt install git-lfs

# import transformers
# print(transformers.__version__)
```

- Nine classification tasks under GLUE Benchmark
- Model checkpoint - weights that will be loaded in a given architecture
- Architecture – the skeleton of the model
- Batch size

```
GLUE_TASKS = ["cola", "mnli", "mnli-mm", "mrpc", "qnli", "qqp", "rte", "sst2", "stsb", "wnli"]
# task = "sst2"
task = "cola"
model_checkpoint = "distilbert-base-uncased"
batch_size = 16
```

Loading the dataset

- load_dataset
- load_metric

```
#Loading the dataset
from datasets import load_dataset, load_metric
actual_task = "mnli" if task == "mnli-mm" else task
dataset = load_dataset("glue", actual_task)
metric = load_metric('glue', actual_task)
```

```
dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['sentence', 'label', 'idx'],
        num_rows: 8551
    })
    validation: Dataset({
        features: ['sentence', 'label', 'idx'],
        num_rows: 1043
    })
    test: Dataset({
        features: ['sentence', 'label', 'idx'],
        num_rows: 1063
    })
})
```

```
dataset["train"][0:3]
```

```
{'sentence': ["Our friends won't buy this analysis, let alone the next one we propose.",
  "One more pseudo generalization and I'm giving up.",
  "One more pseudo generalization or I'm giving up."],
 'label': [1, 1, 1],
 'idx': [0, 1, 2]}
```

Loading the dataset

- show_random_elements function – pick random elements in the data set

```
def show_random_elements(dataset, num_examples=10):
    assert num_examples <= len(dataset), "Can't pick more elements than there are in the dataset."
    picks = []
    for _ in range(num_examples):
        pick = random.randint(0, len(dataset)-1)
        while pick in picks:
            pick = random.randint(0, len(dataset)-1)
        picks.append(pick)

    df = pd.DataFrame(dataset[picks])
    for column, typ in dataset.features.items():
        if isinstance(typ, datasets.ClassLabel):
            df[column] = df[column].transform(lambda i: typ.names[i])
    display(HTML(df.to_html()))
    print(df)

show_random_elements(dataset["train"])
```

	sentence	label	idx
0	I read that Bill had seen myself.	unacceptable	1193
1	Sally might be pregnant, and I believe the claim that Sheila definitely is pregnant.	acceptable	1727
2	The socks are ready for you to announce that you will put on.	unacceptable	1802
3	John loaded the truck with bricks.	acceptable	616
4	Amanda drove the package to New York.	acceptable	2700
5	Danced extremely, Anson frantically at Trade	unacceptable	8283
6	John is prouder of having gone than John expected nobody to believe he would be.	unacceptable	1541
7	We caught them eaten the bananas.	unacceptable	4009
8	Duty made them never miss the weekly meetings.	acceptable	4490
9	Mary always has preferred lemons to limes.	acceptable	1076

Loading the dataset

- metric – arguments(prediction,reference)

```
import numpy as np

fake_preds = np.random.randint(0, 2, size=(12,))
fake_labels = np.random.randint(0, 2, size=(12,))
print(fake_preds)
print(fake_labels)
metric.compute(predictions=fake_preds, references=fake_labels)
```

```
[1 1 0 0 1 0 0 1 0 1 1 0]
[1 1 1 1 0 0 1 0 1 0 0 1]
{'matthews_correlation': -0.50709255283711}
```

- for CoLA: Matthews Correlation Coefficient
- for MNLI (matched or mismatched): Accuracy
- for MRPC: Accuracy and F1 score
- for QNLI: Accuracy
- for QQP: Accuracy and F1 score
- for RTE: Accuracy
- for SST-2: Accuracy
- for STS-B: Pearson Correlation Coefficient and Spearman's_Rank_Correlation_Coefficient
- for WNLI: Accuracy

Preprocessing the data

- Tokenizer checkpoint should be the same as the model checkpoint
- Subword tokenization
- Correspondence of task to column names

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained(model_checkpoint, use_fast=True)
tokenizer("Hello, this one sentence!", "And this sentence goes with it.")
```

```
tokenizer("Hello, this one sentence!")
```

```
{'input_ids': [101, 7592, 1010, 2023, 2028, 6251, 999, 102], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1]}
```

```
task_to_keys = {
    "cola": ("sentence", None),
    "mnli": ("premise", "hypothesis"),
    "mnli-mm": ("premise", "hypothesis"),
    "mrpc": ("sentence1", "sentence2"),
    "qnli": ("question", "sentence"),
    "qqp": ("question1", "question2"),
    "rte": ("sentence1", "sentence2"),
    "sst2": ("sentence", None),
    "stsb": ("sentence1", "sentence2"),
    "wnli": ("sentence1", "sentence2"),
}
sentence1_key, sentence2_key = task_to_keys[task]
if sentence2_key is None:
    print(f"Sentence: {dataset['train'][0][sentence1_key]}")
else:
    print(f"Sentence 1: {dataset['train'][0][sentence1_key]}")
    print(f"Sentence 2: {dataset['train'][0][sentence2_key]}")
```

Preprocessing the data

- preprocess_function – abstraction of different GLUE task inputs to encoded dataset

```python
def preprocess_function(examples):
    if sentence2_key is None:
        return tokenizer(examples[sentence1_key], truncation=True)
    return tokenizer(examples[sentence1_key], examples[sentence2_key], truncation=True)

encoded_dataset = dataset.map(preprocess_function, batched=True)
```

```python
encoded_dataset["train"][0]
```

```
{'sentence': "Our friends won't buy this analysis, let alone the next one we propose.",
 'label': 1,
 'idx': 0,
 'input_ids': [101,
  2256,
  2814,
  2180,
  1005,
  1056,
  4965,
  2023,
  4106,
  1010,
  2292,
  2894,
  1996,
  2279,
  2028,
  2057,
  16599,
  1012,
  102],
 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

```python
dataset["train"][0]
```

```
{'sentence': "Our friends won't buy this analysis, let alone the next one we propose.",
 'label': 1,
 'idx': 0}
```

Fine-tuning the model

- AutoModel class and all of its relatives are actually simple wrappers over the wide variety of models available in the library.
- num_labels - Number of labels to use in the last layer added to the model, typically for a classification task
- metric name
- Training Arguments

```
from transformers import AutoModelForSequenceClassification, TrainingArguments, Trainer

num_labels = 3 if task.startswith("mnli") else 1 if task=="stsb" else 2

model = AutoModelForSequenceClassification.from_pretrained(model_checkpoint, num_labels=num_labels)

metric_name = "pearson" if task == "stsb" else "matthews_correlation" if task == "cola" else "accuracy"
model_name = model_checkpoint.split("/")[-1]
```

```
args = TrainingArguments(
    f"{model_name}-finetuned-{task}",
    evaluation_strategy = "epoch",
    save_strategy = "epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=5,
    weight_decay=0.01,
    load_best_model_at_end=True,
    metric_for_best_model=metric_name,
    #push_to_hub=True,
)
```

Fine-tuning the model

- Compute metrics
- Data collator

```
def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    if task != "stsb":
        predictions = np.argmax(predictions, axis=1)
    else:
        predictions = predictions[:, 0]
    return metric.compute(predictions=predictions, references=labels)

from transformers import DataCollatorWithPadding
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

validation_key = "validation_mismatched" if task == "mnli-mm" else
"validation_matched" if task == "mnli" else "validation"

trainer = Trainer(
    model,
    args,
    train_dataset=encoded_dataset["train"],
    eval_dataset=encoded_dataset[validation_key],
    data_collator=data_collator,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)

trainer.train()
trainer.evaluate()
trainer.push_to_hub()
```

Results

- aaronryanmanuel/distilbert-base-uncased-finetuned-cola

Training Loss	Epoch	Step	Validation Loss	Matthews Correlation
0.5367	1.0	535	0.5657	0.3638
0.3692	2.0	1070	0.5291	0.4912
0.2503	3.0	1605	0.5442	0.5038
0.1895	4.0	2140	0.7376	0.5112
0.1363	5.0	2675	0.8233	0.5229

- aaronryanmanuel/distilbert-base-uncased-finetuned-sst2

Training Loss	Epoch	Step	Validation Loss	Accuracy
0.1877	1.0	4210	0.3477	0.9060
0.1272	2.0	8420	0.3760	0.9094
0.1	3.0	12630	0.3727	0.9106
0.0629	4.0	16840	0.4434	0.9060
0.0369	5.0	21050	0.5134	0.9025

Hyperparameter search

- Data collator

```python
def my_hp_space(trial):
    return {
        "learning_rate": trial.suggest_float("learning_rate", 5e-5, 1e-2, log=True),
        "num_train_epochs": trial.suggest_int("num_train_epochs", 1, 5),
        "seed": trial.suggest_int("seed", 20, 38),
        "per_device_train_batch_size": trial.suggest_categorical("per_device_train_batch_size", [16, 32, 64]),
    }

def model_init():
    return AutoModelForSequenceClassification.from_pretrained(model_checkpoint, num_labels=num_labels)

trainer = Trainer(
    model_init=model_init,
    args=args,
    train_dataset = encoded_dataset["train"].shard(index=1, num_shards=10),
    eval_dataset=encoded_dataset[validation_key],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)
best_run = trainer.hyperparameter_search(n_trials=10, direction="maximize", hp_space=my_hp_space)

for n, v in best_run.hyperparameters.items():
    setattr(trainer.args, n, v)

trainer.train()
trainer.evaluate()
trainer.push_to_hub()
```

```
best_run
```

```
BestRun(run_id='2', objective=0.32493888732834025, hyperparameters={'learning_rate': 1.4198387916440833e-05, 'num_train_epochs': 5, 'seed': 10,
'per_device_train_batch_size': 16})
```