AutoGluon.Multimodal 1.4.0

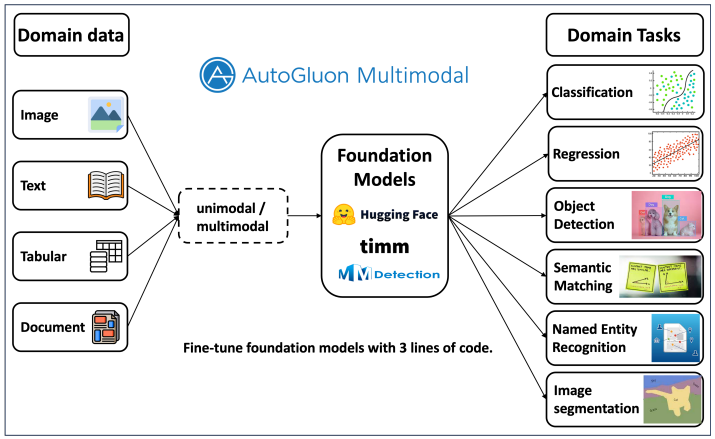
Installation

AutoGluon (GitHub) requires pip > 1.4 (upgrade by pip install -U pip). More installation options. AutoGluon supports Python 3.9 to 3.12. Installation is available for Linux, MacOS, and Windows.

```
pip install autogluon
```

Import Autogluon Multimodal:

```
from autogluon.multimodal import MultiModalPredictor
```



Classification & Regression

MultiModalPredictor finetunes foundation models for solving classification and regression problems with image, text, and tabular features. Here, we use a simplified version petfinder_for_tutorial from the PetFinder dataset. MultiModalPredictor automatically analyzes the columns in the input dataframe to detect categorical, numerical, text, and images (stored as paths or bytearrays).

```
import pandas as pd
train_data = pd.read_csv('train.csv', index_col=0)
test_data = pd.read_csv('test.csv', index_col=0)
```

To train the model, just call '.fit()'. We also support customization (docs).

```
predictor = MultiModalPredictor(
    problem_type="classification",
    label="AdoptionSpeed"
)
predictor.fit(train_data)
```

To extract embedding, or evaluate/inference on the test set.

```
predictor.extract_embedding(test_data)
predictor.evaluate(test_data)  # Evaluation
predictor.predict(test_data)  # Inference
# To Predict probability
predictor.predict_proba(test_data)
```

Object Detection

To use MultiModalPredictor for object detection, please first install additional dependencies by

```
mim install "mmcv==2.1.0"
pip install "mmdet==3.2.0"
pip install "mmengine>=0.10.6"
```

MultiModalPredictor supports common object detection data formats such as COCO (recommended) and VOC. Here we use the dataset tiny_motorbike_coco to demonstrate how to use MultiModalPredictor. The predictor natively supports json files in the COCO-format. We can also visualize the detected bounding boxes with its confidence scores,

```
train_path = "./Annotations/trainval_cocoformat.json"
test_path = "./Annotations/test_cocoformat.json"
predictor = MultiModalPredictor(
    problem_type="object_detection",
    sample_data_path=train_path,
)
predictor.fit(train_path) # Train the detector
predictor.evaluate(test_path) # Evaluation
predictor.predict(test_path) # Inference
```

Named-Entity Recognition

MultiModalPredictor supports named-entity recognition. We use MIT movies corpus to demonstrate the usage, which can be downloaded from train.csv and test.csv.

```
import pandas as pd
predictor = MultiModalPredictor(
    problem_type="ner", label="entity_annotations"
)
# Train model
predictor.fit(pd.read_csv("train.csv"))
# Evaluation
predictor.evaluate(pd.read_csv("test.csv"))
# Inference
text = "Game of Thrones is an American fantasy "
       "drama TV series created by David Benioff"
pred = predictor.predict('text_snippet': [text]})
```

Image Segmentation

While Segment Anything Model (SAM) performs exceptionally well on generic scenes, it encounters challenges when applied to specialized domains like manufacturing, agriculture, etc. MultiModalPredictor mitigates the issue by fine-tuning on domain specific data. Below is an example on Leaf Disease Segmentation dataset.

```
import pandas as pd
train_data = pd.read_csv('train.csv', index_col=0)
test_data = pd.read_csv('test.csv', index_col=0)
image_col, label_col = 'image', 'label'
predictor = MultiModalPredictor(
    problem_type="semantic_segmentation",
    label=label_col,
)
# Finetuning (optional, zero-shot is also supported)
predictor.fit(train_data=train_data)
```

Under the hood, we use LoRA for efficient fine-tuning. Note that, without hyperparameter customization, the huge SAM serves as the default model, which requires efficient fine-tuning in many cases. After fine-tuning, evaluate/predict on the test data.

```
predictor.evaluate(test_data, metrics=["iou"])
pred = predictor.predict(test_data)
```

We can visualize the image, the predicted mask before and after fine-tuning (docs).

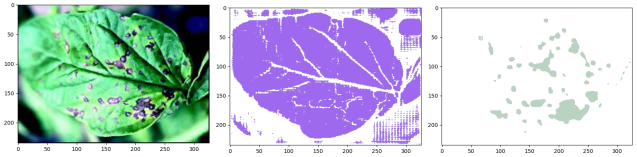


image SAM SAM+PEFT

As evident from the results, the predicted mask after finetuning is much closer to the groundtruth. This demonstrates the effectiveness of using MultiModalPredictor to fine-tune SAM for domain-specific applications, enhancing its performance in tasks like leaf disease segmentation

Semantic Matching

MultiModalPredictor implements a flexible twin-tower architecture that can solve text-text, image-image, and text-image matching problems (docs). Here is an example of finetuning the matching model via relevance data, demonstrated via the Flickr30K image-text matching dataset preprocessed in the dataframe format: flickr30k.zip.

```
import pandas as pd
train_data = pd.read_csv("train.csv", index_col=0)
tdata = pd.read_csv("test.csv", index_col=0)
```

To finetune model, just specify the "query" and "response" keys when creating predictor and pick "image_text_similarity" as problem type.

```
predictor = MultiModalPredictor(
            query="caption",
            response="image",
            problem_type="image_text_similarity",
        )
# Finetuning (optional, zero-shot is also supported)
predictor.fit(train_data, time_limit=180)
# Extract embedding
e_i = predictor.extract_embedding(tdata["image"])
e_t = predictor.extract_embedding(tdata["caption"])
```

- MultiModalPredictor also supports model and hyperparameter customization (docs), knowledge distillation (docs), few shot learning (docs), parameter-efficient finetuning (docs), HPO (docs), and more.
- For deployment, check AutoGluon-Cloud.
- For other use-cases, check TabularPredictor and TimeSeriesPredictor.
- Check the latest version of this cheat sheet.
- Any questions? Ask here
- Like what you see? Consider starring AutoGluon on GitHub and following us on twitter to get notified of the latest updates!