Assignment4

12112124 尹晨

1.What is conformal prediction?

"Conformal prediction" is a method in the field of statistics and machine learning that aims to provide reliable predictions and estimates of their forecast uncertainties. Unlike traditional point forecasting methods, compliance forecasting provides a range of confidence in the forecast, making it more suitable for applications where uncertainty is required.

The basic idea of compliance forecasting is to turn the prediction problem into a compliance problem in which the confidence interval or probability of the prediction is determined by the distribution of the data and the model. The goal of this method is to ensure that predictions of future observations are accurate at a given confidence level.

A key concept in compliance forecasting is the "conformity level", which represents the level of confidence in the forecast. Specifically, for a given level of compliance, a compliance forecast generates a range within which there is a probability that the predicted value will equal or exceed the compliance level. The advantage of this approach is that it does not rely on specific assumptions about the distribution of data, so it is more versatile when faced with different types of data and problems.

One application area for compliance forecasting is anomaly detection and credit scoring. In these areas, accurate estimates of forecast reliability and uncertainty are important. By providing confidence intervals or probability estimates, compliance forecasting methods provide decision-makers with more comprehensive information to help them better understand and utilize forecast outcomes.

2. Code example

```
# 手动设置命令行参数
class Args:
    def __init__(self, seed, alpha, predictor, score, penalty, kreg, weight,
split):
        self.seed = seed
        self.alpha = alpha
        self.predictor = predictor
        self.score = score
        self.penalty = penalty
        self.kreg = kreg
        self.weight = weight
        self.split = split

args = Args(seed=0, alpha=0.1, predictor="Standard", score="THR", penalty=1,
kreg=0, weight=0.2, split="random")

fix_randomness(seed=args.seed)

model_name = 'ResNet101'

# load model
model = torchvision.models.resnet101(weights="IMAGENET1K_V1", progress=True)
```

```python
model_device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(model_device)

# load dataset (Using CIFAR-10)
transform = trn.Compose([trn.Resize(256),
                         trn.CenterCrop(224),
                         trn.ToTensor(),
                         trn.Normalize(mean=[0.485, 0.456, 0.406],
                                       std=[0.229, 0.224, 0.225])
                         ])
dataset = torchvision.datasets.CIFAR10(root=os.path.join(os.path.expanduser('~'),
"data"), train=False, download=True, transform=transform)

cal_dataset, test_dataset = torch.utils.data.random_split(dataset, [5000, 5000])
 # 根据需要调整划分
cal_data_loader = torch.utils.data.DataLoader(cal_dataset, batch_size=64,
shuffle=False, pin_memory=True)
test_data_loader = torch.utils.data.DataLoader(test_dataset, batch_size=64,
shuffle=False, pin_memory=True)

alpha = args.alpha
print(f"Experiment--Data : CIFAR-10, Model : {model_name}, Score : {args.score},
Predictor : {args.predictor}, Alpha : {alpha}")

num_classes = 1000
if args.score == "THR":
    score_function = THR()
elif args.score == "APS":
    score_function = APS()
elif args.score == "RAPS":
    score_function = RAPS(args.penalty, args.kreg)
elif args.score == "SAPS":
    score_function = SAPS(weight=args.weight)
else:
    raise NotImplementedError

if args.predictor == "Standard":
    predictor = SplitPredictor(score_function, model)
elif args.predictor == "ClassWise":
    predictor = ClassWisePredictor(score_function, model)
elif args.predictor == "Cluster":
    predictor = ClusterPredictor(score_function, model, args.seed)
else:
    raise NotImplementedError
print(f"The size of calibration set is {len(cal_dataset)}.")
predictor.calibrate(cal_data_loader, alpha)
# predictor.evaluate(test_data_loader)

# test examples
print("Testing examples...")
prediction_sets = []
labels_list = []
with torch.no_grad():
    for examples in tqdm(test_data_loader):
        tmp_x, tmp_label = examples[0], examples[1]
        prediction_sets_batch = predictor.predict(tmp_x)
        prediction_sets.extend(prediction_sets_batch)
        labels_list.append(tmp_label)
test_labels = torch.cat(labels_list)

metrics = Metrics()
print("Etestuating prediction sets...")
print(f"Coverage_rate: {metrics('coverage_rate')(prediction_sets,
test_labels)}.")
print(f"Average_size: {metrics('average_size')(prediction_sets, test_labels)}.")
print(f"CovGap: {metrics('CovGap')(prediction_sets, test_labels, alpha,
num_classes)}.")
```

3.Output meaning

The final output of the model consists of three parts that are used to test conformal prediction effect:

Coverage Rate:

This metric measures the proportion of samples in the prediction sets that contain real labels.

Formula: Coverage = Number of samples with real labels/total number of test set samples.

Explanation: The higher the coverage, the better, because it means that the model's prediction set is more likely to contain the true label, that is, the model's prediction of the sample is more accurate.

Average Size:

The average size represents the average number of samples for each prediction set.

Formula: Average size = average of the number of samples in all prediction sets.
Explanation: The smaller the average size, the better, because it means that each prediction set is more compact and the model's uncertainty about the sample is more finely considered.

CovGap (Coverage Gap) :

The coverage gap is the difference between coverage and desired coverage. The expected coverage is calculated based on the alpha parameter, which represents the expected coverage at a certain confidence level.

Formula: Coverage gap = Coverage - expected coverage.

Explanation: The smaller the coverage gap, the better, because it indicates that the model's predictions are more consistent with the desired coverage. If the coverage gap is large, it may indicate that the model's predictions are overly optimistic or conservative.

4. Workflow

Call the TorchCP library example, pytorch built-in database (CIFAR - 10, CIFAR - 100, MINST, FashionMINST, SVHN) for conformal prediction.

The code example in the second part only shows the procedure for calling CIFAR-10.

The final output of the three metrics mentioned in Part 3 is used to test the effect of conformal prediction.

5. Result of different database

CIFAR-10

```
Experiment--Data : CIFAR-10, Model : ResNet101, Score : THR, Predictor :
Standard, Alpha : 0.1
The size of calibration set is 5000.
Testing examples...
100%|██████████| 79/79 [00:10<00:00,  7.56it/s]
Etestuating prediction sets...
Coverage_rate: 0.8788.
Average_size: 911.9408.
CovGap: 9.948091821157124.
```

CIFAR-100

```
Experiment--Data : CIFAR-100, Model : ResNet101, Score : THR, Predictor :
Standard, Alpha : 0.1
The size of calibration set is 5000.
Testing examples...
100%|██████████| 79/79 [00:09<00:00,  8.21it/s]
Evaluating prediction sets...
Coverage_rate: 0.8914.
Average_size: 922.2136.
CovGap: 10.340776907910568.
```

MINST

```
Experiment--Data : MNIST, Model : ResNet101, Score : THR, Predictor : Standard,
Alpha : 0.1
The size of calibration set is 5000.
Testing examples...
100%|██████████| 79/79 [00:09<00:00,  8.44it/s]
Evaluating prediction sets...
Coverage_rate: 0.9032.
Average_size: 942.0134.
CovGap: 10.83743783916796.
```

FashionMINST

```
Experiment--Data : FashionMNIST, Model : ResNet101, Score : THR, Predictor :
Standard, Alpha : 0.1
The size of calibration set is 5000.
Testing examples...
100%|██████████| 79/79 [00:09<00:00,  8.52it/s]
Evaluating prediction sets...
Coverage_rate: 0.7498.
Average_size: 894.687.
CovGap: 20.33783188070897.
```

SVHN

```
Experiment--Data : SVHN, Model : ResNet101, Score : THR, Predictor : Standard,
Alpha : 0.1
The size of calibration set is 13016.
Testing examples...
100%|██████████| 204/204 [00:30<00:00,  6.66it/s]
Evaluating prediction sets...
Coverage_rate: 0.8999692685925015.
Average_size: 957.0096035648432.
CovGap: 10.822547382309603.
```

6 Effect analysis

CIFAR-10

The CIFAR-10 dataset, using the ResNet101 model, is evaluated with the following indicators:

Coverage Rate: 0.8788, indicating that 87.88% of the samples in the forecast set contain true labels.
Average Size: 911.9408, meaning that the average sample size for each prediction set is about 911.
Coverage Gap (CovGap) : 9.9481, representing the difference between the coverage rate and the desired coverage rate of 9.9481.

Preliminary analysis:

The high coverage (87.88%) indicates that the model's prediction set contains the true label relatively well, which is a positive sign.
The average size is 911, which can be a relatively large value and may mean that the size of the prediction set is large. In some cases, this may be considered a high level of uncertainty.
The coverage gap is 9.9481, which is large and may indicate that there is room for improvement in the calibration of the model, as it shows a large gap between actual and expected coverage.

Suggestions for improvement:

If the high average size is considered a problem, try adjusting the parameters of the model or algorithm to reduce the size of the prediction set.
A large coverage gap may indicate that the model needs to be better calibrated, and consideration may be given to trying different plasticity prediction algorithms, adjusting model parameters, or further calibrating the prediction set to reduce the gap.

CIFAR-100

The CIFAR-100 dataset, using the ResNet101 model, is evaluated with the following indicators:

Coverage Rate: 0.8914, indicating that 89.14% of the samples in the prediction set contain true labels.
Average Size: 922.2136, meaning that the average sample size for each prediction set is about 922.
Coverage Gap (CovGap) : 10.3408, representing the difference between the coverage rate and the desired coverage rate of 10.3408.

Preliminary analysis:

Compared to CIFAR-10, CIFAR-100 had a slightly higher coverage rate of 89.14%, which is a positive sign that the model also performed well on more fine-grained classification tasks.
A larger average size may indicate that the model has higher uncertainty for some samples, resulting in a larger set of predictions.

The coverage gap of 10.3408 is still large, which may indicate that there is still room for improvement in the calibration of the model.

Suggestions for improvement:

If the high average size is considered to be a problem, consider adjusting the parameters of the model or algorithm to reduce the size of the prediction set.
Further optimization of the calibration of the model is considered to reduce the coverage gap.

MINST

The MNIST dataset, using the ResNet101 model, is evaluated with the following indicators:

Coverage Rate: 0.9032, which means that 90.32% of the samples in the prediction set contain true labels.
Average Size: 942.0134, meaning that the average sample size for each prediction set is about 942.
Coverage Gap (CovGap) : 10.8374, representing the difference between the coverage rate and the desired coverage rate of 10.8374.

Preliminary analysis:

For the MNIST dataset, the coverage was high, reaching 90.32%, which is a positive sign that the model performed well for the classification task of handwritten digits.
A larger average size may indicate that the model has higher uncertainty for some samples, resulting in a larger set of predictions.
The coverage gap of 10.8374 is still large, which may indicate that the calibration of the model still has room for improvement.

Suggestions for improvement:

If the high average size is considered to be a problem, consider adjusting the parameters of the model or algorithm to reduce the size of the prediction set.
Further optimization of the calibration of the model is considered to reduce the coverage gap.

FashionMINST

The FashionMNIST dataset, using the ResNet101 model, is evaluated on the following metrics:

Coverage Rate: 0.7498, indicating that 74.98% of the samples in the prediction set contain true labels.
Average Size: 894.687, meaning that the average sample size for each prediction set is about 895.
Coverage Gap (CovGap) : 20.3378, representing the difference between coverage and desired coverage of 20.3378.

Preliminary analysis:

The relatively low coverage (74.98%) may indicate that the model is relatively cautious in its predictions on the FashionMNIST dataset, with fewer samples included in the prediction set.
The relatively large average size may indicate that the model has higher uncertainty for some samples, resulting in a larger set of predictions.
The coverage gap of 20.3378 is large, which may indicate that the calibration of the model has great room for improvement.

Suggestions for improvement:

If relatively low coverage is considered a problem, consider adjusting the parameters of the model or algorithm to improve the inclusivity of the forecast set.
If the high average size is considered to be a problem, consider adjusting the parameters of the model or algorithm to reduce the size of the prediction set.
Further optimization of the calibration of the model is considered to reduce the coverage gap.

SVHN

The SVHN dataset, using the ResNet101 model, is evaluated with the following indicators:

Coverage Rate: 0.89997, indicating that 89.997% of the samples in the prediction set contain true labels.
Average Size: 957.0096, meaning that the average sample size for each prediction set is about 957.
Coverage Gap (CovGap) : 10.8225, representing the difference between the coverage rate and the desired coverage rate of 10.8225.

Preliminary analysis:

Coverage is relatively high (89.997%), which is a positive sign that the model's predictions on the SVHN dataset are relatively accurate.
The relatively large average size may indicate that the model has higher uncertainty for some samples, resulting in a larger set of predictions.
The coverage gap of 10.8225 is large, which may indicate that there is room for improvement in the calibration of the model.

Suggestions for improvement:

If the high average size is considered to be a problem, consider adjusting the parameters of the model or algorithm to reduce the size of the prediction set.
Further optimization of the calibration of the model is considered to reduce the coverage gap.

Comparison of results

From the results provided, the following is a summary of the evaluation indicators compared to the various data sets:

Coverage Rate:

Highest: MNIST dataset, 90.32%
Minimum: FashionMNIST dataset, 74.98%

Average Size:

Minimum: FashionMNIST dataset, 894.687
Max: MNIST dataset, 942.0134

Coverage Gap (CovGap) :

Minimum: CIFAR-10 dataset, 9.9481
Max: FashionMNIST dataset, 20.3378

Judgment result:

The dataset that works best: the MNIST dataset, because it performs best in coverage, while also having relatively small average size and coverage gaps.

The least effective dataset: the FashionMNIST dataset because it performs poorly in coverage, with a larger average size and a larger coverage gap.

Possible reasons for poor results:
FashionMNIST dataset: Low coverage and high average size may indicate that the model's predictions on this dataset are relatively conservative and that there is a high degree of uncertainty for some samples. The FashionMNIST dataset may contain some of the more challenging samples, making it difficult for models to predict accurately. Solutions may include more complex models, tweaking model parameters, or using algorithms that are better suited to the data set.

7. Deficiencies and improvements

Based on the experimental results provided, we can preliminarily speculate the possible shortcomings and aspects that need to be improved in TorchCP library:

Calibration performance: There is a relatively large coverage gap across different data sets, which may indicate that the model has room for improvement in calibration performance. Further optimization of the calibration algorithm may be required to improve the calibration of the model for different data sets.

Average size: On some datasets, the average size is relatively large, which may indicate that the model has higher uncertainty for some samples, resulting in a larger set of predictions. The algorithm can be improved to make the prediction set more compact while maintaining good calibration.

Applicability: Large performance differences between different data sets may indicate that current algorithms perform inconsistently when dealing with different types of data sets. Further improvements to the algorithm may be needed to make TorchCP perform better on a wider range of data sets.

Algorithm parameters: More detailed experiments and analysis are needed to determine the effect of the model's hyperparameters on different data sets. Tuning algorithm parameters can have a positive effect on performance.

Interpretability and user friendliness: Does the TorchCP library provide enough interpretative information for users to understand the model's prediction set and calibration performance? In practical use, users may need more detailed information to adjust the model or explain the model's behavior.

It should be noted that these are only preliminary guesses based on the output results, and the actual direction of improvement may require more in-depth experiments and analysis.