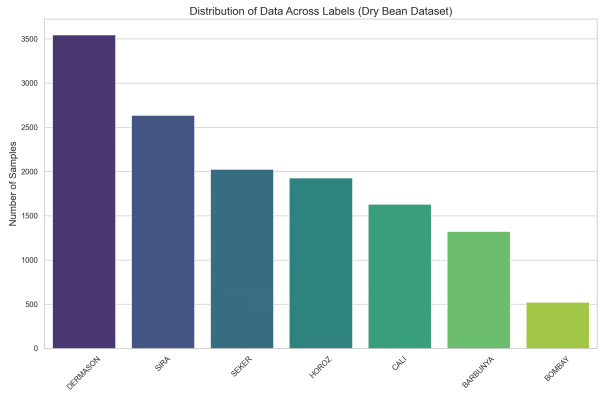
[A3] Classification with Reject Option

2021313075 백경인

1. 데이터셋

https://archive.ics.uci.edu/dataset/602/dry+bean+dataset 에서 받아온 데이터로, 등록된 7 종의 건조콩 1361(# of Data point)알의 이미지를 고해상도 카메라로 촬영했습니다. 총 16 가지 feature 들로 12 개의 치수와 4 개의 모양 형태가 입자로부터 얻어졌습니다. 7 종의 Label 의 분포는 아래와 같다.



2. 인공신경망 학습을 위한 최적의 데이터 전처리 방법 및 하이퍼파라미터 설정 제시

아무래도 전처리 방법은 관례적인 data scaling, label encoding 등 했습니다. 하이퍼 파라미터 설정에 큰 노력을 했습니다. Validation set 을 설정했습니다. 교수님의 교안에도 적혀있듯이 하이퍼파라미터 설정할 때에는 validation set 을 기반으로 설정하라 가 생각나서 아무래도 uncertainty threshold 설정을 할 때에 validation 을 20% 설정했습니다. Optimizing threshold 과정을 거쳤습니다. 각 uncertainty mehtod 에 따라 값의 범위가 다양하기에 어떠한 threshold 을 설정하냐에 따라 마무리 단계에서 성능차이를 일으킬 것이라고 생각했습니다. 이로써 각 method 들의 minimum 값과 maximum 값을 100 개의 동등한 간격으로 나누어 최고의 성능을 줄 수 있는 값으로 threshold 을 설정했습니다.

3. 3 개 이상의 불확실성 정량화(Uncertainty quantification) 방법 비교와 Test set 에서 예측 불확실성이 높은 data point 의 제외에 따른 성능 개선 정도 분석

Accuracy without reject option: 92.1043 || Best Margin Threshold: -0.9899, Accuracy: 93.0060, Rejection Rate: 54.26%

Best Confidence Threshold: -0.9728, Accuracy: 92.9761, Rejection Rate: 37.55% || Best Entropy Threshold: 0.0135, Accuracy: 93.9815, Rejection Rate: 64.29%

아무래도 rejection rate 을 떠나 accuracy 가 가장 높은 모델과 threshold 에 집중했습니다. 하지만 rejection rate 이 최대 64%나 되는 것을 보고 다시 생각의 필요성을 느꼈습니다. 그 이유는 reject option 을 설정하지 않았을 때, 92.1043 와 큰 차이가 없기에 rejection 이 필요한가를 가장 생각했습니다. 그랬을 때, confidence measure 의 특성상 최고 확률값의 음수라고 생각했을때, threshold 가 -0.9728 이라고 한다면, 97.28% 확률값 이하의 값들은 reject 을 하겠다인데, 97.28% 값 정도는 사실 불확실성이 없는 것이라고 생각이 들어 아무래도 동등하게 나누기보다 min 과 max 의 평균값 or 중간값으로 threshold 을 고정했습니다.

4. 추가 성능 개선을 위한 방안 논의

HW3 에서 저에겐 추가적인 성능 개선에 대한 방법을 고민하는 것이 매력적이게 다가오지 못했습니다. 모델을 Neural network 으로 설정한 이상 black-box 이기에 해석적인 관점을 중요시 하려했습니다. 해석력이 높아진 모델이 항상 성능향상을 유도하진 않기 때문이라고 생각했습니다.

Data 가 어떻게 얻어졌으며 이에 대한 meta-data 를 알고 있기에 data feature 을 잘 이용하면 좋겠다고 생각했습니다. 우선 Data point 들은 알의 이미지를 고해상도 카메라로 촬영했으며 12 개의 치수와 4 개의 모양 형태가 입자로부터 얻어졌습니다. 이로써 12 개의 치수 feature 를 모아 하나의 모델을 만들고 4 개의 모양 형태 feature 를 모아 또 하나의 모델을 만들어 두 모델을 병합하는 방식을 생각했습니다.

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import numpy as np
import pandas as pd
from ucimlrepo import fetch_ucirepo  # Import the dataset fetching function
import matplotlib.pyplot as plt
import seaborn as sns
```

Python

1. Fetch Dry Bean Dataset using ucimlrepo

```python
dry_bean = fetch_ucirepo(id=602)
```

Python

Data (features and targets as pandas dataframes)

```python
X = dry_bean.data.features
y = dry_bean.data.targets
```

Python

Handle missing values by dropping rows with missing values

```python
X = X.dropna()
y = y.loc[X.index]  # Ensure targets match the remaining data points
```

Python

Encode class labels as integers

```python
le = LabelEncoder()
y = le.fit_transform(y)
```

Python

Split data into training, validation, and test sets (60% train, 20% val, 20% test)

```python
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=42)  # 60% train
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)  # 20% val, 20% test
```

Python

Standardize the features

```python
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

Python

Convert the data to PyTorch tensors

```python
X_train = torch.tensor(X_train, dtype=torch.float32)
X_val = torch.tensor(X_val, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
y_val = torch.tensor(y_val, dtype=torch.long)
y_test = torch.tensor(y_test, dtype=torch.long)
```

Python

Create DataLoader for PyTorch

```python
batch_size = 32
train_dataset = TensorDataset(X_train, y_train)
val_dataset = TensorDataset(X_val, y_val)
test_dataset = TensorDataset(X_test, y_test)
```

Python

```python
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

Python

2. Define Neural Network Model in PyTorch

```python
class NeuralNet(nn.Module):
    def __init__(self, input_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, 64)
        self.dropout = nn.Dropout(0.5)  # Dropout layer for MC Dropout
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, num_classes)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.dropout(x)
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Python

Initialize the model, loss function, and optimizer

```python
input_size = X_train.shape[1]
num_classes = len(np.unique(y_train))
model = NeuralNet(input_size, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Python

3. Training Function

```python
def train(model, train_loader, criterion, optimizer, epochs=50):
    model.train()
    for epoch in range(epochs):
        running_loss = 0.0
        for i, (inputs, labels) in enumerate(train_loader):
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        print(f"Epoch [{epoch+1}/{epochs}], Loss: {running_loss/len(train_loader):.4f}")
```

Python

4. Evaluation Function

```python
def evaluate(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = 100 * correct / total
    return accuracy
```

Python

Train the model

```python
train(model, train_loader, criterion, optimizer, epochs=512)
```

Python

Evaluate without reject option

```python
accuracy = evaluate(model, test_loader)
print(f"Accuracy without reject option: {accuracy:.4f}")
```

Python

5. Uncertainty Quantification

1. Confidence Uncertainty Measure

```python
def confidence_uncertainty(softmax_probs):
    max_probs, _ = torch.max(softmax_probs, dim=1)
    return -max_probs
```

Python

2. Margin Uncertainty Measure

```python
def margin_uncertainty(softmax_probs):
    sorted_probs, _ = torch.sort(softmax_probs, dim=1, descending=True)
    margin = sorted_probs[:, 0] - sorted_probs[:, 1]
    return -margin
```

Python

3. Entropy Uncertainty Measure

```python
def entropy_uncertainty(softmax_probs):
    entropy = -torch.sum(softmax_probs * torch.log(softmax_probs + 1e-10), dim=1)  # Add small epsilon for numerical stability
    return entropy
```

Python

Get predictions and calculate uncertainties

```python
def calculate_uncertainties(model, test_loader):
    model.eval()
    confidence_list = []
    margin_list = []
    entropy_list = []

    with torch.no_grad():
        for inputs, _ in test_loader:
            outputs = model(inputs)
            softmax_probs = F.softmax(outputs, dim=1)

            # Calculate uncertainties
            confidence_list.append(confidence_uncertainty(softmax_probs))
            margin_list.append(margin_uncertainty(softmax_probs))
            entropy_list.append(entropy_uncertainty(softmax_probs))

    # Convert lists to tensors
    confidence_uncertainty_tensor = torch.cat(confidence_list)
    margin_uncertainty_tensor = torch.cat(margin_list)
    entropy_uncertainty_tensor = torch.cat(entropy_list)

    return confidence_uncertainty_tensor, margin_uncertainty_tensor, entropy_uncertainty_tensor
```

Python

Run uncertainty calculations

```python
confidence_uncertainty_vals, margin_uncertainty_vals, entropy_uncertainty_vals = calculate_uncertainties(model, test_loader)
```

Python

For example, let's print the first few uncertainty values

```python
print("Confidence Uncertainty (first 5):", confidence_uncertainty_vals[:5])
print("Margin Uncertainty (first 5):", margin_uncertainty_vals[:5])
print("Entropy Uncertainty (first 5):", entropy_uncertainty_vals[:5])
```

Python

Function to test different thresholds for a given uncertainty measure

```python
def optimize_threshold(uncertainty_vals, y_true, model, val_loader, num_thresholds=100):
    # Create thresholds between the min and max of uncertainty values
    range_min = uncertainty_vals.min().item()
    range_max = uncertainty_vals.max().item()

    thresholds = torch.linspace(range_min, range_max, num_thresholds)

    best_threshold = None
    best_accuracy = 0.0
    best_rejection_rate = 0.0
    for threshold in thresholds:
        accepted_idx = uncertainty_vals <= threshold  # Accept if uncertainty is below threshold
        rejected_idx = uncertainty_vals > threshold   # Reject if uncertainty is above threshold

        # Calculate accuracy for accepted samples
        accepted_samples = torch.nonzero(accepted_idx, as_tuple=True)[0]
        y_accepted = y_true[accepted_samples]
        if len(accepted_samples) > 0:
            correct = 0
            total = 0
            for i, (inputs, labels) in enumerate(val_loader):
                if i in accepted_samples:
                    outputs = model(inputs)
                    _, predicted = torch.max(outputs, 1)
                    total += labels.size(0)
                    correct += (predicted == labels).sum().item()
            accuracy = 100 * correct / total if total > 0 else 0.0
        else:
            accuracy = 0.0
        rejection_rate = 100 * (1 - len(accepted_samples) / len(uncertainty_vals))
```

```python
        # Choose the best threshold based on accuracy
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_threshold = threshold
            best_rejection_rate = rejection_rate
    return best_threshold, best_accuracy, best_rejection_rate
```

Python

Apply the optimization for each uncertainty measure

```python
def find_best_thresholds(model, val_loader, y_val):
    confidence_uncertainty_vals, margin_uncertainty_vals, entropy_uncertainty_vals = calculate_uncertainties(model, val_loader)

    print("Optimizing thresholds for uncertainty measures...")

    # Optimize thresholds for each uncertainty measure
    best_conf_threshold, best_conf_accuracy, best_conf_rejection_rate = optimize_threshold(confidence_uncertainty_vals, y_val, model, val_loader)
    best_margin_threshold, best_margin_accuracy, best_margin_rejection_rate = optimize_threshold(margin_uncertainty_vals, y_val, model, val_loader)
    best_entropy_threshold, best_entropy_accuracy, best_entropy_rejection_rate = optimize_threshold(entropy_uncertainty_vals, y_val, model, val_loader)
    print(f"Best Confidence Threshold: {best_conf_threshold:.4f}, Accuracy: {best_conf_accuracy:.4f}, Rejection Rate: {best_conf_rejection_rate:.2f}%")
    print(f"Best Margin Threshold: {best_margin_threshold:.4f}, Accuracy: {best_margin_accuracy:.4f}, Rejection Rate: {best_margin_rejection_rate:.2f}%")
    print(f"Best Entropy Threshold: {best_entropy_threshold:.4f}, Accuracy: {best_entropy_accuracy:.4f}, Rejection Rate: {best_entropy_rejection_rate:.2f}%")
```

Python

Run the threshold optimization on the validation set

```python
find_best_thresholds(model, val_loader, y_val)
```

Python