XAI Exercises

Felix Neubürger

2025

Exercise Sheet 1

Learning Objectives 1

- Understand basic concepts of interpretable machine learning
- Implement simple interpretable models
- Analyze model coefficients and feature importance

Exercise 1: Linear Model Interpretation (Basics)

Resources: Chapter 5.1 of textbook, Slides 15-28

1. Train a linear regression model on the Boston housing dataset
2. Extract and interpret the model coefficients
3. Calculate feature importance using standardized coefficients
4. Compare with Lasso regression results (feature selection)

```
# Starter code
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression, Lasso

data = load_boston()
X, y = data.data, data.target
# Add your implementation here
```

Exercise 2: Decision Tree Analysis

Resources: Chapter 5.5 of textbook, Slides 45-52

1. Train a decision tree classifier on the iris dataset
2. Visualize the decision tree using graphviz
3. Extract and interpret feature importance values
4. Modify tree depth and analyze impact on interpretability

Exercise 3: Model Comparison

Resources: Chapter 6 of textbook, Slides 60-65

Compare the interpretability of:

- Linear regression vs. decision tree
- Global vs. local explanations
- Model-specific vs model-agnostic methods

Submission Guidelines

- Submit Jupyter notebooks with implementations and markdown explanations
- Include visualizations of model interpretations

Recommended Resources

- LIME in Practice
- scikit-learn Documentation

Exercise Sheet 2

Exercise 1: Basic Feature Importance (45 mins)

Dataset: Wine Quality (UCI)
`https://archive.ics.uci.edu/dataset/186/wine+quality`
Tools: sklearn, SHAP

1. Load dataset and train Random Forest (15 mins)

   ```python
   from sklearn.ensemble import RandomForestClassifier
   # Starter code for data loading
   ```

2. Calculate permutation importance (15 mins)

3. Create SHAP summary plot and interpret the results (15 mins)

   ```python
   import shap
   explainer = shap.TreeExplainer(model)
   shap_values = explainer.shap_values(X_test)
   ```

Exercise 2: Partial Dependence Plots (45 mins)

Dataset: Breast Cancer Wisconsin (UCI)
`https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic`

1. Create PDP for 2 features using sklearn (15 mins)

   ```python
   from sklearn.inspection import PartialDependenceDisplay
   ```

2. Interpret results and identify interactions. Discuss with the class (30 mins)

Theoretical Questions (Bonus)

- Why might feature importance differ between permutation and SHAP?
- When would PDP be preferred over ICE plots?

Exercise 3

Exercise 1: Grad-CAM with TorchCAM (45 mins)

Tools: torchcam, torchvision
Use the package documentation for reference:
`https://frgfm.github.io/torch-cam/`

1. Setup environment and load model (15 mins)

```
import torch
from torchcam.methods import GradCAM
from torchvision.models import resnet18
from torchvision.datasets import CIFAR10

# Load pretrained model
model = resnet18(pretrained=True)
model.fc = torch.nn.Linear(512, 10)  # Adapt for CIFAR-10
model.load_state_dict(torch.load('cifar10_resnet18.pth'))
```

2. Generate and visualize explanations (30 mins)

3. Play around with the layers you are using for the explanations.

4. What are the differences?

```
from torchcam.utils import overlay_mask
from PIL import Image

# Initialize Grad-CAM
cam_extractor = GradCAM(model, 'layer4')

# Process sample image
img = Image.open("sample_airplane.jpg")
inputs = transform(img).unsqueeze(0)

# Generate heatmap
out = model(inputs)
activation_map = cam_extractor(out.squeeze(0).argmax().item(), out)
```

```python
# Overlay on image
result = overlay_mask(img, activation_map[0], alpha=0.5)
result.show()
```

Pre-configured Setup

```bash
# Recommended environment
conda create -n xai python=3.8
conda install pytorch torchvision -c pytorch
pip install torchcam captum quantus pillow
```

Sample Solutions Checklist

- Working Grad-CAM visualizations for 3 classes
- Faithfulness scores for both methods
- Comparative analysis table

Exercise 4

Explainable AI: CNN Interpretability with CIFAR-10 (Pre-built Tools)

Exercise 1: Comparative Analysis

Packages: torchcam vs captum

1. Generate explanations with both torchcam and captum
2. Generate explanations with other methods available in the captum and torchcam library `https://github.com/pytorch/captum`

   ```
   # Captum implementation
   from captum.attr import LayerGradCam

   gradcam = LayerGradCam(model, model.layer4)
   attr = gradcam.attribute(inputs, target=out.argmax())
   ```

3. Compare results using Quantus `https://github.com/understandable-machine-intelligence-lab/Quantus`

   ```
   from quantus import FaithfulnessCorrelation

   metric = FaithfulnessCorrelation()
   scores = metric(model, inputs, out.argmax(),
                  [activation_map.numpy(), attr.detach().numpy()])
   ```

Key Questions

- Which package provides more intuitive visualizations?
- How do computational costs compare?
- Which method better highlights discriminative features?

Pre-configured Setup

```
# Recommended environment
conda create -n xai python=3.8
conda install pytorch torchvision -c pytorch
pip install torchcam captum quantus pillow
```

Sample Solutions Checklist

- Working Grad-CAM visualizations for 3 classes
- Faithfulness scores for both methods
- Comparative analysis table

Exercise Sheet 5

Exercise 1: Basic Attention Visualization

Tools: BERTviz & HuggingFace Transformers

1. Setup environment and load model

   ```
   pip install bertviz transformers
   ```

2. Visualize attention in different layers

   ```python
   from bertviz import head_view
   from transformers import BertModel, BertTokenizer

   model = BertModel.from_pretrained('bert-base-uncased', output_attentions=True)
   tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

   text = "The cat sat on the mat because it was tired."
   inputs = tokenizer.encode(text, return_tensors='pt')
   outputs = model(inputs)
   attention = outputs[-1]  # Tuple of attention tensors

   # Visualize
   head_view(attention, tokenizer.convert_ids_to_tokens(inputs[0]))
   ```

3. Tasks:
   - Identify attention heads focusing on pronouns ("it")
   - Compare patterns in early vs. final layers
   - Find heads specializing in syntactic vs semantic relationships

Exercise 2: Feature Attribution with Captum

Resources: Captum BERT Tutorial

1. Implement integrated gradients

```
from captum.attr import IntegratedGradients

ig = IntegratedGradients(model)
attributions = ig.attribute(inputs, target=0,
                            n_steps=50,
                            return_convergence_delta=True)
```

2. Compare with attention weights

```
# Visualize both side-by-side
import matplotlib.pyplot as plt
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.imshow(attention[0][0].detach().numpy())
ax2.imshow(attributions[0].sum(dim=-1).detach().numpy())
```

Theoretical Questions

- How does attention visualization differ from feature attribution?
- What are the limitations of visualizing attention as explanation?
- When would you prefer attention visualization over gradient-based methods?

Advanced Tasks (Bonus)

- Modify visualization for multi-head comparison
- Calculate attention head importance scores
- Implement custom attention pattern filtering

Troubleshooting Tips

- If BERTviz doesn't render: Use Jupyter notebook directly
- For CUDA memory issues: Reduce model size to ‘bert-tiny‘
- Version conflicts: Pin to ‘transformers==4.25.1‘, ‘bertviz==1.0.0‘

Recommended Resources

- BERTviz GitHub
- Captum BERT Tutorial
- Attention is not Explanation (Paper)

Exercise 6

Explainable AI: LLM Interpretability with Llama2

Exercise 1: Setup & Baseline Attribution

Tools: Captum, HuggingFace Transformers

1. Load a pretrained LLM with huggingface that fits your GPU
2. if no GPU is available use the `https://www.ki.fh-swf.de/` server
3. Llama-3.1-8B `https://huggingface.co/meta-llama/Llama-3.1-8B`
4. Llama-3.2-1B `https://huggingface.co/meta-llama/Llama-3.2-1B`
5. Llama-3.2-1B `https://huggingface.co/meta-llama/Llama-3.2-3B`

```
from transformers import AutoTokenizer, AutoModelForCausalLM
model = AutoModelForCausalLM.from_pretrained("/meta-llama/Llama-3.1-8B")
tokenizer = AutoTokenizer.from_pretrained("/meta-llama/Llama-3.1-8B")
```

6. Generate baseline predictions

```
prompt = "Explain the concept of quantum entanglement:"
inputs = tokenizer(prompt, return_tensors="pt")
outputs = model.generate(inputs.input_ids, max_length=200)
print(tokenizer.decode(outputs[0]))
```

Exercise 2: Feature Attribution

Method: Layer Integrated Gradients

1. Implement attribution calculation

```
from captum.attr import LayerIntegratedGradients

def attribute(model, inputs):
    lig = LayerIntegratedGradients(model, model.model.layers[0])
```

```python
    attributions = lig.attribute(
        inputs=inputs.input_ids,
        baselines=tokenizer("", return_tensors="pt").input_ids,
        target=outputs
    )
    return attributions
```

2. Visualize token importance

```python
import numpy as np
import matplotlib.pyplot as plt

tokens = tokenizer.convert_ids_to_tokens(inputs.input_ids[0])
attr_scores = np.sum(attributions[0].detach().numpy(), axis=1)

plt.barh(tokens, attr_scores)
plt.title("Token Attribution Scores")
plt.show()
```

Exercise 3: Comparative Analysis

- Compare different attributions following the https://captum.ai/tutorials/Llama2_LLM_Attribution :
- Discuss stability of attributions

Theoretical Questions

1. Why use layer-specific attribution instead of full-model?
2. How does choice of baseline impact IG results?
3. What challenges arise when interpreting auto-regressive models?

Optimization Tips

- Use `model.half()` for FP16 precision
- Limit sequence length to 512 tokens
- Cache model weights with `model.cache=True`
- Use `return_convergence_delta=True` for reliability checks

Advanced Tasks (Bonus)

- Implement attention head visualization yourself
- Compare with `FeatureAblation` method

Recommended Resources

- Official Captum Tutorial
- Llama2 Paper
- Llama2 GitHub