Integrated Gradients Research Report

Author: Sheng Wang a1903948
Team: RAIL-PG-2

1. Introduction

Railway track prediction is a critical task aimed at forecasting potential failures or damages based on input features like acceleration, track curvature, load-bearing capacity, and speed. Accurate predictions are essential for ensuring safety and efficient railway operations. However, a key challenge is ensuring the transparency and interpretability of these prediction models. Understanding why a model makes certain predictions is vital for trust and decision-making.

To achieve this, I research attribution methods that can highlight the contributions of different input features in the model's decision-making process. A good attribution method must satisfy two key criteria:

- Sensitivity: The attribution must change when the input features change
- Implementation Invariance: The method should produce consistent results regardless of model structure.

Many previous attribution methods fall short of meeting both of these requirements simultaneously. In contrast, Integrated Gradients (IG) has emerged as a robust technique that satisfies both the sensitivity and implementation invariance criteria(Sundararajan, Taly, & Yan, 2017). This method offers a clear way to interpret deep learning models, especially when working with complex datasets like those in railway track prediction.

In the following sections, I will briefly outline the principles of Integrated Gradients, its role in model interpretability, its limitations, and evaluate how well it fits with the specific needs of the railway track prediction project.

2. Method Overview: Integrated Gradients

Integrated Gradients (IG) is a widely used attribution method for explaining deep learning models. It provides an explanation of the model's predictions by calculating the contribution of input features to the output.

IG has two key characteristics(Sundararajan, Taly, & Yan, 2017):

- Implementation Invariance: It is independent of the model architecture.
- Sensitivity: The attribution results change when the input features change.

The IG method evaluates the contribution of input features by calculating the gradient integral along the straight-line path between the input and the baseline.

Formally, suppose we have a function$F: \Re^n \rightarrow [0,1]$ representing a deep network, where $x \subset \Re^n$ is the input and $x' \subset \Re^n$is the baseline input. For image networks, the baseline could be a black image, and for text models, it could be the zero embedding vector.
We consider the straight-line path from the baseline $\boldsymbol{x}'$ to the input $\boldsymbol{x}$, computing gradients along this path. Integrated gradients are obtained by accumulating these gradients, which can be interpreted as the path integral of gradients along this straight line from $\boldsymbol{x}'$ to $\boldsymbol{x}$.
For an input $\boldsymbol{x}$ and baseline $\boldsymbol{x}'$, the integrated gradient along the i-th dimension is defined as(Sundararajan, Taly, & Yan, 2017):

$$\text{IntegratedGrads}_i(x) := (x_i - x_i') \times \int_0^1 \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i} d\alpha$$

Figure1. Integrated Gradient Equation

This method allows us to compute the contribution of each input feature to the final model prediction, which is fundamental for model interpretability.

3. Applicability of Integrated Gradients

The Group Lasso Transformer is a powerful model that performs feature selection through integrated Lasso regression and uses the Transformer architecture to handle complex sequential data. Since Integrated Gradients (IG) is applicable to many deep learning models, including Transformer-based models, it is theoretically suitable for the Group Lasso Transformer.

3.1. Applicability

1. Model Type
    - The **Group Lasso Transformer** is based on the Transformer architecture, which is a deep neural network. IG can be used for attribution analysis in such models. The core idea of IG is to calculate the contribution of input features to the prediction output, making it suitable for explaining complex models like the Transformer.
2. Attribution Analysis Requirements

- For the **Group Lasso Transformer**, you might want to understand how the model makes predictions based on different features, feature groups, or input data at various time steps. IG can provide an analysis of these feature contributions.

3. Limitations of IG

- Computational Overhead

  Although the calculation process of IG is relatively straightforward, it may still require significant computational resources for large deep learning models. In the railway track prediction task, if the data size is large, IG could introduce additional computational burden.

- Sensitivity to Baseline Selection

  IG is sensitive to the choice of baseline. If the baseline is not appropriately chosen, the attribution results may be distorted. For example, selecting an unreasonable baseline image or an inappropriate input feature baseline could lead to incorrect attributions.

- Challenges with Time-Series Data

  Railway track prediction may involve time-series data, which requires IG to effectively handle temporal dependencies. Although IG is not specifically designed for time-series data, with appropriate processing (e.g., feature engineering), it can still be applied to time-series data.

3.2 How to Perform Attribution Analysis with IG in Code

1. Install Necessary Libraries
   %pip install captum

2. Prepare the Model and Data

```
# ========== 1. Load Data from Table ==========
print("Step 1: Load Feature Data from Table")
TABLE =
"`09ad024f-822f-48e4-9d9e-b5e03c1839a2`.feature_selection.preprocess_training_table"
df = spark.table(TABLE)

# Filter all numeric features starting with Wagon_
```

```
numeric_types = ("double", "float", "int", "bigint", "decimal")
feature_cols = [
    c for c, t in df.dtypes
    if c.startswith("Wagon_")
    and c != "Wagon_RecordingDate"
    and any(tp in t.lower() for tp in numeric_types)
]

# Select required columns
selected_cols = ["Tc_BaseCode", "Tc_SectionBreakStartKM", "Tc_date"] +
feature_cols
df_selected = df.select(*selected_cols)

# Handle missing values (fill with 0)
for col in feature_cols:
    df_selected = df_selected.withColumn(col, F.coalesce(F.col(col),
F.lit(0.0)))

# ========== 2. Convert to Pandas DataFrame ==========
print("Step 2: Convert to Pandas DataFrame")
sample_size = 100
df_pandas = df_selected.limit(sample_size).toPandas()

# ========== 3. Load Trained Model ==========
print("Step 3: Load Trained Model")
model_path = './model.pth'
model = torch.load(model_path)
model.eval()

# ========== 4. Select a Sample for Attribution Analysis ==========
print("Step 4: Perform Attribution Analysis on Single Sample")
# Select the first sample
sample_idx = 0
sample_info = df_pandas.iloc[sample_idx][["Tc_BaseCode",
"Tc_SectionBreakStartKM", "Tc_date"]]
sample_features = df_pandas.iloc[sample_idx][feature_cols].values

print(f"\nSample Information:")
print(f"  Base Code: {sample_info['Tc_BaseCode']}")
print(f"  Section Break Start KM:
{sample_info['Tc_SectionBreakStartKM']}")
```

```
print(f"  Date: {sample_info['Tc_date']}")# Convert to tensor
input_tensor = torch.FloatTensor(sample_features).unsqueeze(0)  # [1,
num_features]

# ========== 5. Model Prediction ==========
with torch.no_grad():
    probabilities = model(input_tensor)
    prediction = torch.argmax(probabilities, dim=1)

print(f"\nModel Prediction Results:")
print(f"  Predicted Class: {'Fracture ⚠' if prediction.item() == 1
else 'No Fracture ✓'}")
print(f"  No Fracture Probability: {probabilities[0][0].item():.2%}")
print(f"  Fracture Probability: {probabilities[0][1].item():.2%}")

# ========== 6. Integrated Gradients Attribution Analysis ==========
print("Step 6: Perform Integrated Gradients Attribution Analysis")
def forward_func(inputs):
    """Returns fracture probability"""
    probs = model(inputs)
    return probs[:, 1]
```

3. Initialize Integrated Gradients

```
# Initialize IG
ig = IntegratedGradients(forward_func)

# Create baseline
baseline = torch.zeros_like(input_tensor)

print("Computing attribution scores...")

# Calculate attributions
attributions = ig.attribute(
    input_tensor,
    baselines=baseline,
    n_steps=50
)
attributions_np = attributions.squeeze().detach().numpy()
print("Attribution computation completed!\n")
```

```
# ========== 7. Attribution Results Analysis ==========
print("Step 7: Attribution Results Analysis")

# Create attribution DataFrame
attribution_df = pd.DataFrame({
   'Feature Name': feature_cols,
   'Feature Value': sample_features,
   'Attribution Score': attributions_np,
   'Absolute Attribution': np.abs(attributions_np),
   'Impact Direction': ['Promotes Fracture' if x > 0 else 'Reduces
Fracture' for x in attributions_np]
})

# Sort by absolute attribution
attribution_df = attribution_df.sort_values('Absolute Attribution',
ascending=False)
attribution_df['Rank'] = range(1, len(attribution_df) + 1)
attribution_df = attribution_df[['Rank', 'Feature Name', 'Feature
Value', 'Attribution Score',
                                 'Absolute Attribution', 'Impact
Direction']]

# Display Top 10
print(f"Top 10 Most Important Features:")
print(attribution_df.head(10).to_string(index=False))
```

4. Visualize Attribution Results

```
# ========== 8. Visualize Attribution Results ==========
print("Step 8: Visualize Attribution Results")
# Select top 15 most important features
top_n = 15
top_df = attribution_df.head(top_n)

# Create visualization
fig, ax = plt.subplots(figsize=(10, 6))

# Color bars: red for positive attribution, blue for negative
colors = ['red' if x > 0 else 'blue' for x in top_df['Attribution
Score']]

# Horizontal bar chart
```

```
ax.barh(range(len(top_df)), top_df['Attribution Score'], color=colors,
alpha=0.7)
ax.set_yticks(range(len(top_df)))
ax.set_yticklabels(top_df['Feature Name'])
ax.set_xlabel('Attribution Score')
ax.set_title(f'Top {top_n} Most Important Features\n(Red=Promotes
Fracture, Blue=Reduces Fracture)')
ax.axvline(x=0, color='black', linestyle='--')
ax.invert_yaxis()

plt.tight_layout()
plt.savefig('attribution_results.png', dpi=300, bbox_inches='tight')
plt.show()
```

4. Conclusion

Through the theoretical study of Integrated Gradients (IG), analysis of code feasibility, and exploration of its limitations, it is concluded that IG is suitable for the InsightFactory railway prediction project. Below, I will summarize why Integrated Gradients (IG) can be applied to this project from three perspectives: Applicability, Code Implementation, and Interpretability.

- Applicability

  For the Group Lasso Transformer model, Integrated Gradients (IG) is a suitable explanation method as it provides attributions for each input feature, helping to understand which features contribute most to the model's predictions.

- Code Implementation:

  - The Captum library can be used to implement IG and compute the contribution of each feature.
  - Attribution results can then be visualized(Chatzimparmpas et al., 2020), allowing a clearer understanding of the model's decision-making process.

- Interpretability:

  IG helps to interpret how the Group Lasso Transformer model makes predictions when facing a variety of input features. This is particularly useful for understanding the model’s decisions when dealing with complex or grouped features

5. Reference

Sundararajan, M., Taly, A., & Yan, Q. (2017, July). Axiomatic attribution for deep networks. In *International conference on machine learning* (pp. 3319-3328). PMLR.

Chatzimparmpas, A., Martins, R. M., Jusufi, I., & Kerren, A. (2020). A survey of surveys on the use of visualization for interpreting machine learning models. *Information Visualization*, *19*(3), 207-233.