

Kapil Yadav, B22AI024

Bhagwan, B22AI010

Integrated Gradients Explainability for AnnexML Model: Implementation, Results, and Analysis

Introduction

Explainable AI (XAI) is crucial for building trust in complex models like AnnexML. This report details the implementation of the Integrated Gradients (IG) method to explain the predictions of an AnnexML model trained on the IAPRTC-12 dataset. Integrated Gradients is a feature attribution method that assigns importance scores to each feature by accumulating the gradients along a path from a baseline input (usually a zero vector) to the actual input. [Sundararajan, Taly, Yan, 2017]

Dataset and Model

The AnnexML model is trained on the IAPRTC-12 dataset, a multi-label classification dataset. Multi-label classification means each data point can belong to multiple classes simultaneously. The dataset consists of image features extracted using MATLAB, stored in `.mat` format. The AnnexML model itself is not detailed in the notebook, but the report assumes it is a pre-trained model used for prediction.

Implementation Steps

The following steps outline the implementation of Integrated Gradients for explaining the AnnexML model:

1. Data Loading and Preprocessing:
 - a. The initial step involves loading the training and testing data from the `.mat` files using `scipy.io.loadmat`.

- b. The data is then converted into a format suitable for using integrated gradients. The sparse matrices returned by `load_svmlight_file` from `sklearn.datasets` are converted into dense numpy arrays.
- 2. `save_instance_as_svm` Function:
 - a. This function takes a data instance (a NumPy array) and saves it into an SVM-light-style format in a text file. Each feature is formatted as `index:value`, and the features are space-separated. A dummy label of 0 is added to the beginning of the line.
- 3. `annexml_predict_svm` Function:
 - a. This function interfaces with the pre-trained AnnexML model. It takes the path to an SVM-formatted input file and an output file path as input.
 - b. It then executes the AnnexML prediction command-line tool via `subprocess.run`. The command specifies the input file, the output file, and the location of the trained AnnexML model.
 - c. The prediction output is parsed from the resulting file. It reads the prediction file, extracts the prediction values, and normalizes them to create a probability distribution (using simple division by the sum of the predictions). The predictions are in the format: `index:value,...`
- 4. `integrated_gradients_annexml` Function:
 - a. This function implements the Integrated Gradients algorithm for the AnnexML model. It takes a `sample_instance`, a `baseline` (defaulting to a zero vector), the number of `steps` for the approximation, and a small `delta` value.
 - b. It generates `steps + 1` interpolated samples along the linear path from the `baseline` to the `sample_instance`.
 - c. It iterates through the steps, saving each interpolated input to a file, getting predictions from AnnexML using the `annexml_predict_svm` function, and accumulating the attributions for each feature and label. A small delta value is used to prevent division by zero.
 - d. The attributions are calculated based on the difference in predictions between consecutive interpolated inputs, approximating the integral of the gradients. The intermediate predictions are stored in a DataFrame and saved to a CSV file.
 - e. Finally, the function returns the integrated gradients, representing the feature attributions for the input instance.
- 5. Usage Example:
 - a. The notebook picks a sample input from the test dataset `X_test_dense`.
 - b. It calls the `integrated_gradients_annexml` function to compute the feature importances.

Results and Analysis

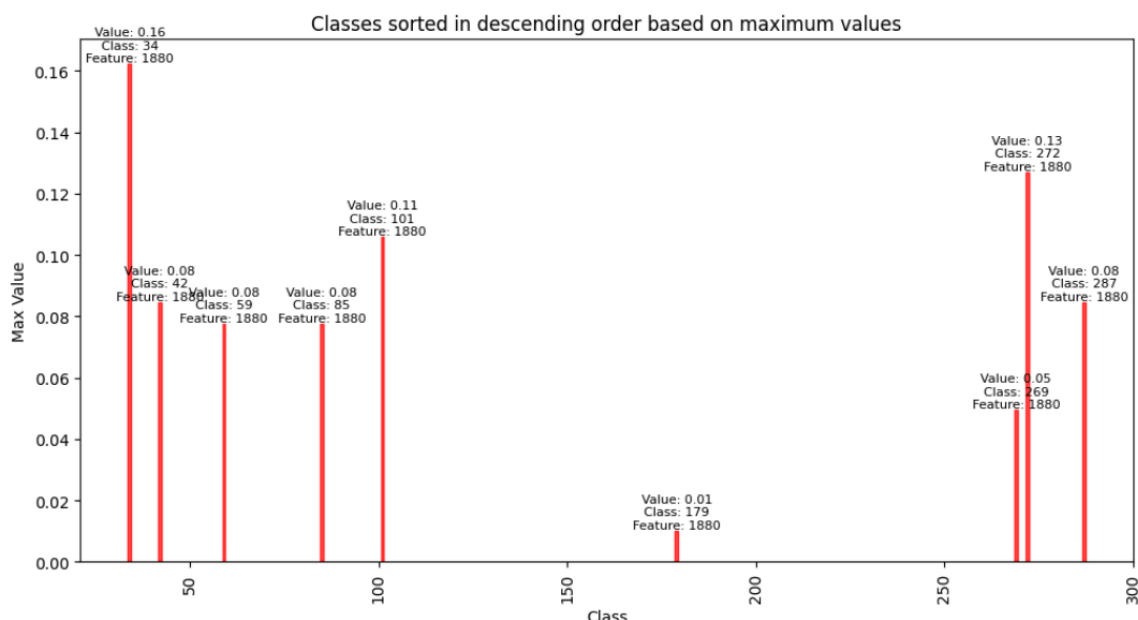
The notebook prints the count of non-zero attributions at each step of the integrated gradients calculation, which remains constant. This is used as a diagnostic, though the value itself isn't directly interpretable.

The notebook also includes code to:

- **Identify Important Features:** The notebook then finds the columns with non-zero attributions in the `feature_importance` matrix. For each of these columns (corresponding to a particular class), it finds the top three rows (corresponding to specific input features) with the highest attribution values. This identifies the features that contribute most to the prediction of that class.
- **Sort Classes by Maximum Feature Importance:** The notebook determines the feature with the largest attribution value for each class, then sorts the classes in descending order based on these maximum values. This provides a ranking of the classes based on the strength of their most influential features.
- **Visualization:** The class values and the corresponding features are printed to the console, and the results are displayed as a bar chart, with the bar values, class numbers, and feature numbers highlighted, in a plot using matplotlib.

For the 1st Test example, classes arranged in descending order according to values they got corresponding to the feature value

```
Classes sorted in descending order based on maximum values:  
Class 34: Max value = 0.1622368202089609, Feature number = 1880  
Class 272: Max value = 0.12696794625049113, Feature number = 1880  
Class 101: Max value = 0.1058066218754093, Feature number = 1880  
Class 42: Max value = 0.08464529750032744, Feature number = 1880  
Class 287: Max value = 0.08464529750032744, Feature number = 1880  
Class 59: Max value = 0.0775915227086335, Feature number = 1880  
Class 85: Max value = 0.0775915227086335, Feature number = 1880  
Class 269: Max value = 0.04937642354185768, Feature number = 1880  
Class 179: Max value = 0.010051629078163885, Feature number = 1880
```



Classes sorted in descending order based on maximum values:

Class 284: Max value = 0.3391313340294602, Feature number = 1606

Class 52: Max value = 0.1403302071846042, Feature number = 1606

Class 212: Max value = 0.09355347145640282, Feature number = 1606

Class 139: Max value = 0.08185928752435245, Feature number = 1606

Class 272: Max value = 0.0701651035923021, Feature number = 1606

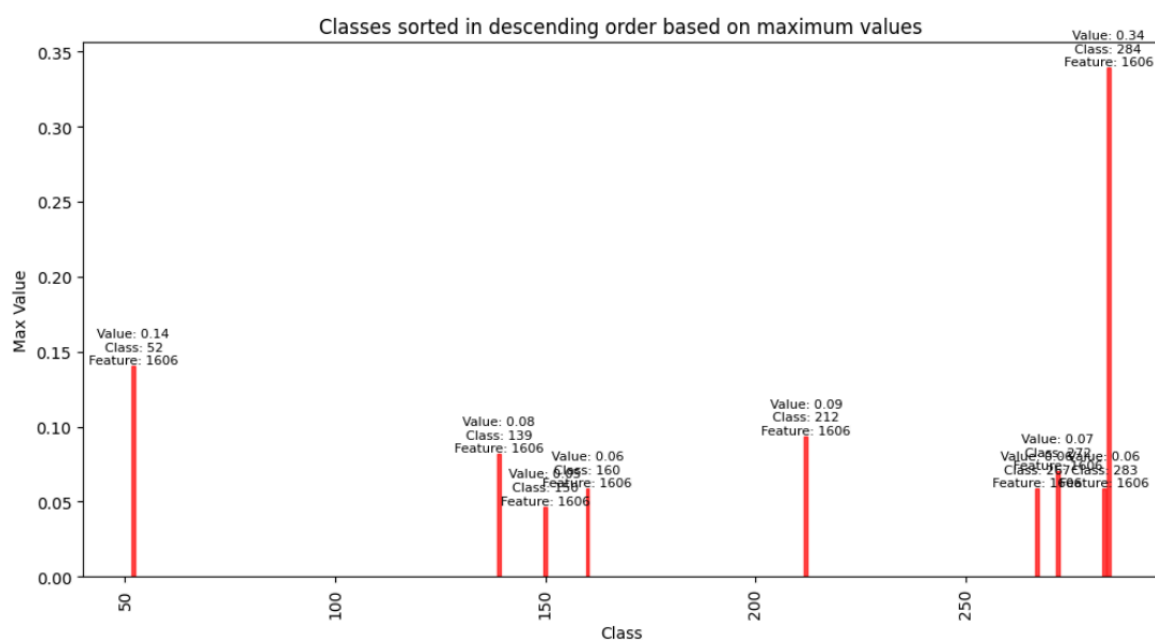
Class 160: Max value = 0.05847091966025176, Feature number = 1606

Class 267: Max value = 0.05847091966025176, Feature number = 1606

Class 283: Max value = 0.05847091966025176, Feature number = 1606

Class 150: Max value = 0.04677673572820141, Feature number = 1606

For the 2nd Example



Discussion

The use of Integrated Gradients provides valuable insights into the AnnexML model's decision-making process. The ability to identify the most influential features for each class, along with a ranking of classes based on feature importance, could be used to improve the model, validate its behavior, or develop trust with users.

Several challenges arise in this implementation:

- AnnexML Interaction: The need to interact with AnnexML via file I/O and command-line calls adds complexity.
- Computational Cost: Integrated Gradients can be computationally expensive, as it requires multiple forward passes through the model. The number of


steps `steps` (set to 50 in this notebook) controls the accuracy of the approximation but also affects the computation time.

- Interpretation of Multi-label Attributions: Although the visualization identifies key features for *each* class, it doesn't directly address the interplay between multiple labels that AnnexML might assign to a single instance.

Conclusion

The provided notebook successfully implements Integrated Gradients to explain the predictions of an AnnexML model. The `integrated_gradients_annexml` function calculates feature attributions, and the subsequent analysis identifies important features for each class and ranks the classes by feature importance. The visualization provides a clear and informative way to understand the model's behavior. Future work could focus on analyzing the interactions between features in the multi-label setting and exploring ways to reduce the computational cost of the algorithm.

Citations

- Sundararajan, M., Taly, A., & Yan, Q. (2017). Axiomatic attribution for deep networks. *Proceedings of the 34th International Conference on Machine Learning*, 70, 3319-3328. <https://arxiv.org/abs/1703.01365>
-  [NeurIPS*2021] Fast Axiomatic Attributions for Neural Networks
- <https://github.com/ankurtaly/Integrated-Gradients>