Anomaly Detection on CelebA Bald vs Non-Bald Dataset
By: Aubrie Pressley & Lisette Kamper-Hinson

1. Task Overview: Anomaly Detection

This project addresses the problem of identifying bald individuals in the CelebA dataset using anomaly detection. Since only 2.2% of examples are labeled as bald, traditional classification models are not well-suited for this task. Instead, anomaly detection models can treat the bald class as rare deviations from the norm and assign anomaly scores. This report compares four models: a dummy baseline, DevNet (deep semi-supervised), Isolation Forest, and KMeans. The motivation for implementing DevNet and comparing it to other state of the art models is described in the literature survey.

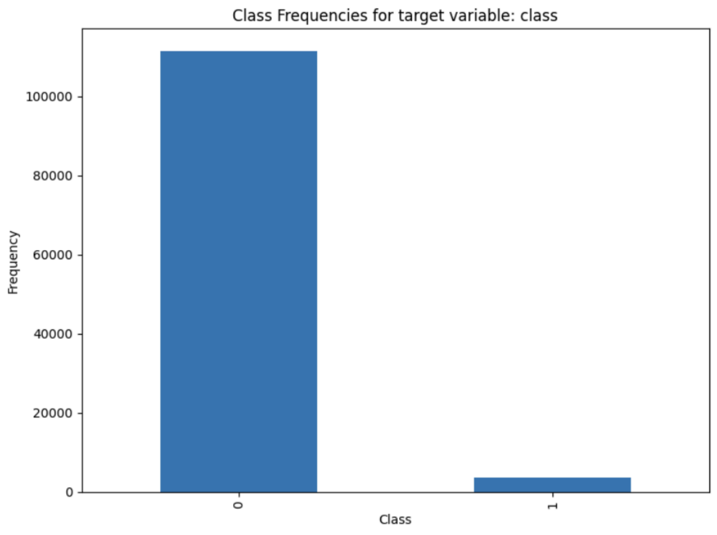
2. Literature Survey

The celebra_baldvsnonbald_normalised.csv (celeba) data was used in the original publication "Deep Anomaly Detection with Deviation Networks" to extend the research of deep learning for anomaly detection. Traditional anomaly detection algorithms are unsupervised due to a lack of known or labeled anomalies. Additionally, traditional algorithms focus on learning new feature representations which can lead to inefficient learning. Unlike the traditional methods, the introduced framework utilizes the few labeled anomalies with neural deviation learning to enhance anomaly scoring [1]. A PyTorch implementation of the framework was developed called DevNet [2]. Experiments involving eight different datasets, including the celeba one, showed that the implementation achieves significant improvements of anomaly detection, data efficiency, and robustness when compared to REPEN, DSVDD, FSNet, and iForest [1].

REPEN, DSVDD, FSNet, and iForest are all current state of the art methods currently employed to study data similar to the celeba dataset. REPEN is good for deep anomaly detection with limited labeled data. DSVDD is good for feature learning in anomaly detection. FSNet is good for few-shot classification. Lastly, iForest is good for unsupervised anomaly detection. However, all four face limitations in performance when labeled anomalies are sparse or when the data distribution is highly complex, which is where DevNet demonstrates significant advantages [1]. The results from our experiment also show that DevNet also outperforms Isolation Forest and KMeans, further supporting the paper's findings.

3. Dataset Description

The dataset consists of 202,599 face image observations with 39 binary features labeled A0 through A38 and a binary target column labeled 'class'. After cleaning, 87,484 duplicate rows were removed. Bald individuals are rare (~2.2%) in the dataset, making it an ideal use case for anomaly detection. Each model was trained on 80% of the cleaned dataset and evaluated on the remaining 20% (development set) using AUC-ROC, Average Precision (AP), and F1 Score.



Class Frequencies for Target Variable

4. Code Design and Implementation Choices

The **`assignment_6.py`** script was designed to be modular, extensible, and evaluation-driven. The core goals were: (1) enable experimentation across multiple anomaly detection models, (2) handle class imbalance with awareness, and (3) output interpretable results for visual and metric-based comparison.

Data Handling and Preprocessing

The dataset is read using a **`utils.read_data()`** function that wraps **`pandas.read_csv()`** for simplicity and clarity. Duplicate rows are removed prior to saving a clean version of the dataset (`celeba_clean.csv`), ensuring model training is not biased by repetition. Data is split into training and development (test) sets using `train_test_split` with a fixed random seed for reproducibility. Since our data mining task was anomaly detection, no transformers were used to remove outliers.

Feature Extraction

Labels and features are separated using **`utils.extract_features()`** which explicitly removes the class column from the feature matrix. All features and labels are cast to integer format **`np.array(...).astype(int)`** to comply with DeepOD's requirements for DevNet and ensure consistency across models.

Model Abstraction and Selection

Each model (DevNet, Isolation Forest, KMeans, Dummy Baseline) is implemented as a separate function. These functions return evaluation metrics (AUC-ROC, Average Precision) and the fitted model object for further use. The model to run is selected using command-line arguments **`--model`** and can include **`devnet`**, **`iforest`**, **`cluster`**, **`baseline`**, or **`all`**.

Evaluation Strategy

All models are evaluated on both training and development sets using:

- **`roc_auc_score()`** – ranking performance
- **`average_precision_score()`** – particularly useful for imbalanced datasets
- **`f1_score()`** – balance between precision and recall using a fixed threshold

For unsupervised models, anomaly predictions are thresholded at the top 2.2% of scores which matches the proportion of anomalies in the dataset).

Score Normalization and Distribution Plots

Anomaly scores are normalized using **`MinMaxScaler()`** for comparability and visualization. Score distribution plots are created using **`matplotlib.pyplot.hist()`** for each model, saved as PNGs, and embedded in the report to visually compare the separation between bald and non-bald instances.

Artificial Data Integration

A second mode **`--type artificial loads`** a synthetic dataset **`artificial_new_data.csv`** and tests the best-trained model from the real data setting. This serves to test the generalizability of the models and highlights how performance degrades when the new data patterns does not match the patterns of the training data.

Model Saving and Reuse

The best model (based on dev AUC) is re-trained on the full cleaned dataset and saved using **`joblib.dump()`** for potential downstream use. This enables repeatable inference and supports the `predict_new_data()` function which demonstrates model portability.

5. Baseline Model (Random Anomaly Assignment)

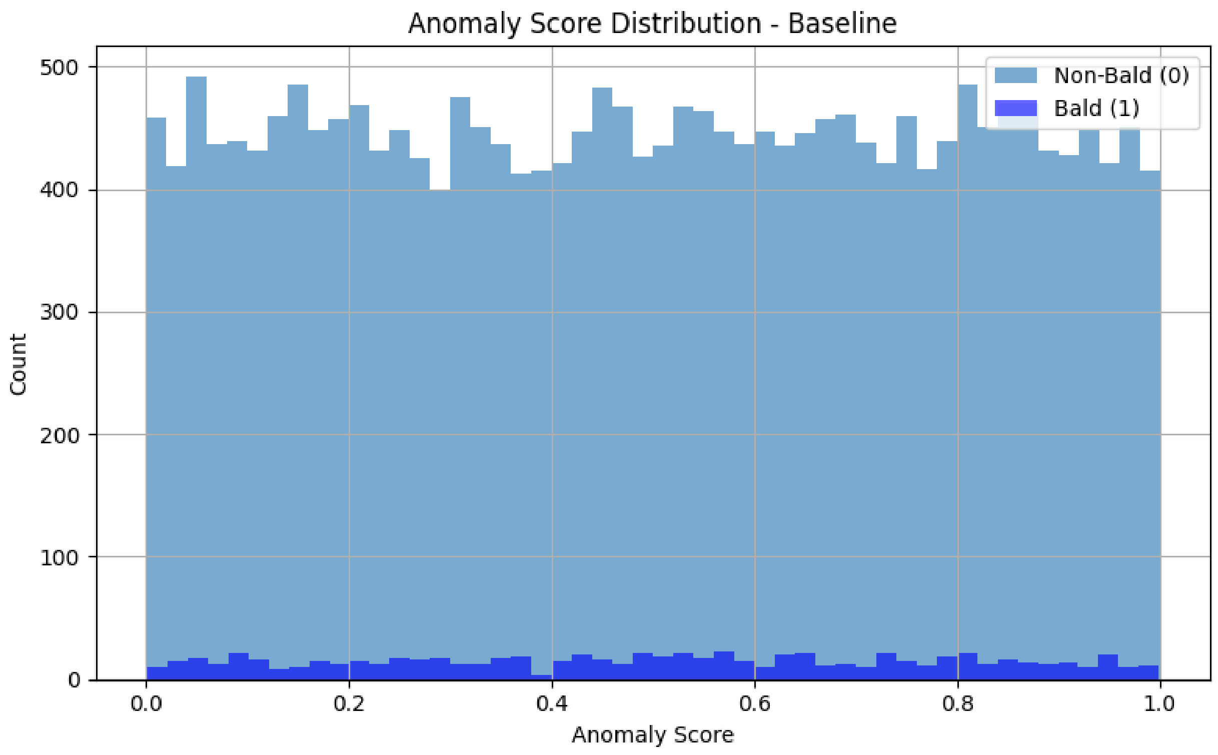
Train Set Evaluation - Dummy (random assignment)

AUC-ROC	Average Precision	F1 Scores
0.4936	0.0297	0.0201

Dev Set Evaluation - Dummy (random assignment)

AUC-ROC	Average Precision	F1 Scores
0.5031	0.0328	0.0205

The baseline model assigns anomaly scores at random, without taking any features into consideration. The purpose of the dummy model is to highlight the performance of DevNet and other state of the art models. Its metrics hover around random guessing thresholds. As expected, its AUC-ROC is near 0.5, and both AP and F1 scores are very low, indicating minimal practical use for detecting bald individuals.



Flat score distribution for the baseline: No visible distinction between bald and non-bald individuals.

6. Anomaly Detection Models

DevNet Setup: Uses labeled anomalies (bald samples) to train a neural network to learn meaningful deviation-based scores.

We were interested in training a model in using DevNet because the authors of the paper “Deep Anomaly Detection with Deviation Networks” demonstrated its ability to achieve strong anomaly detection performance, particularly in imbalance settings [1]. However, the original DevNet implementation was designed for image data and not suitable for tabular data [2]. To apply DevNet to our celeba dataset, we used the DeepOD library [3], which offers a tabular version of DevNet compatible with PyTorch and supports semi-supervised learning.

To successfully run DevNet from DeepOD, the following setup requirements had to be met:

1) Install dependencies via **pip install deepod.**
2) Import DevNet from deepod.models.tabular inside the script file.
3) Ensure compatibility by downgrading NumPy to a version < 2.0, as PyTorch and DeepOD are not yet compatible with NumPy 2.x. Done via **pip install “numpy<2.0” deepod**

An additional challenge to the setup requirements included ensuring all feature and label sets were integers before fitting the DevNet model.

Isolation Forest Setup: An unsupervised tree ensemble model that randomly partitions data to isolate rare cases.

We selected Isolation Forest as a baseline due to its effectiveness in high-dimensional, imbalance settings without requiring labeled anomaly data. It works by constructing random decision trees and scoring samples based on how easily they can be isolated. Anomalies tend to be isolated in fewer splits. Unlike DevNet, Isolation Forest did not present any data formatting challenges, so no special preprocessing beyond scaling was required.

KMeans Setup: An unsupervised clustering model using distance from cluster centers as the anomaly score.

We explored KMeans clustering as a distance-based unsupervised method for anomaly detection. The intuition is that anomalies tend to fall far from dense clusters of normal points. By computing the distance of each sample to its nearest cluster center, we can rank samples based on how anomalous they appear.

One challenge with Kmeans is that K has to be defined before running the model. Therefore, we performed parameter tuning by fitting multiple KMeans models with different values of k (from 2 to 10) on the training set and selecting the model that achieved the best AUC. This allowed us to identify a suitable number of clusters for capturing the normal data structure. Like Isolation Forest, no significant data formatting issues were encountered.

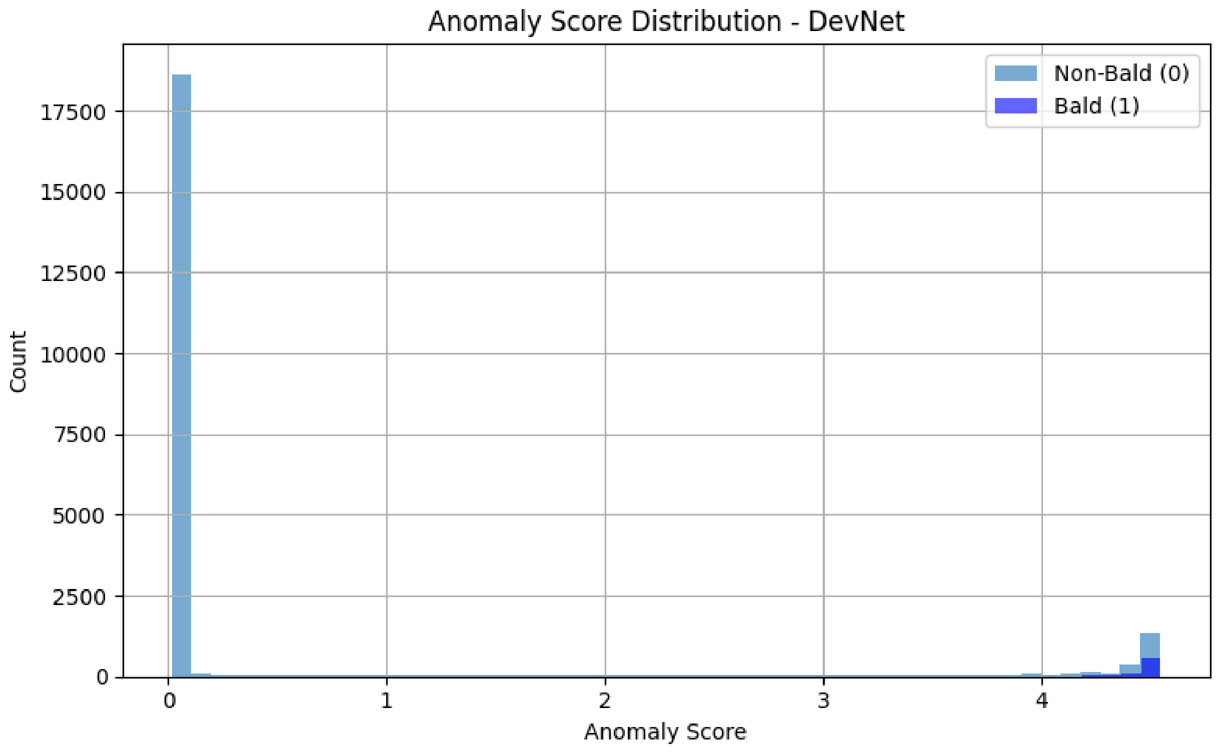
All models output continuous anomaly scores, which are then evaluated using thresholds set at the 97.8th percentile (matching the ~2.2% anomaly rate).

7. DevNet Results

Dev Set Evaluation

AUC-ROC	Average Precision	F1 Scores
0.9588	0.3975	0.3966

DevNet shows exceptional performance. An AUC-ROC of 0.9588 indicates the model is extremely good at ranking bald instances above non-bald ones. Its AP (0.3975) and F1 (0.3966) are much higher than other models, showing that it captures the few bald instances with high confidence and minimal false positives. This is a clear demonstration of the benefit of using labeled anomalies in training. DevNet produces a distinct anomaly score distribution with clear separation.



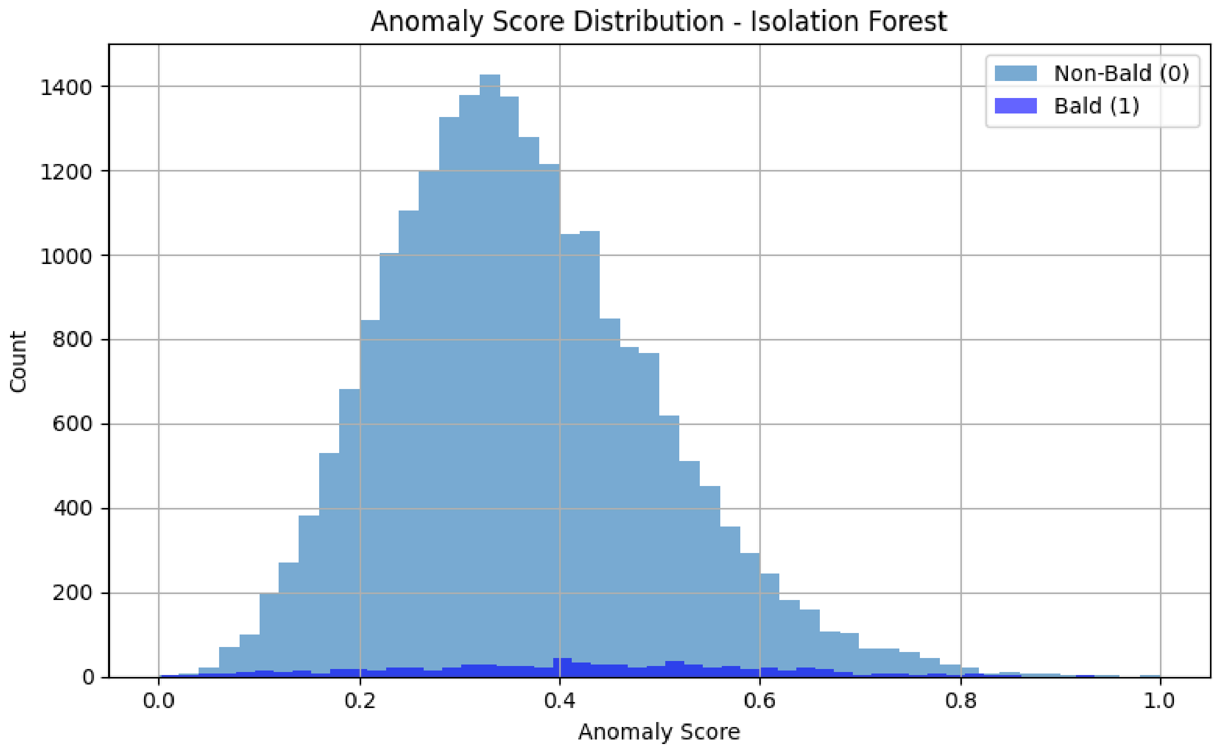
DevNet distribution: non-bald scores cluster near 0 while bald scores cluster separately around 4+.

8. Isolation Forest Results

Dev Set Evaluation

AUC-ROC	Average Precision	F1 Scores
0.5983	0.0591	0.0727

Isolation Forest yields slightly better than random performance, with a bell-shaped score distribution that does not separate anomalies well. Its AUC-ROC of ~0.6 is far below the DevNet's, and both AP and F1 scores are minimal. This reflects the limits of fully unsupervised methods in highly imbalanced scenarios.



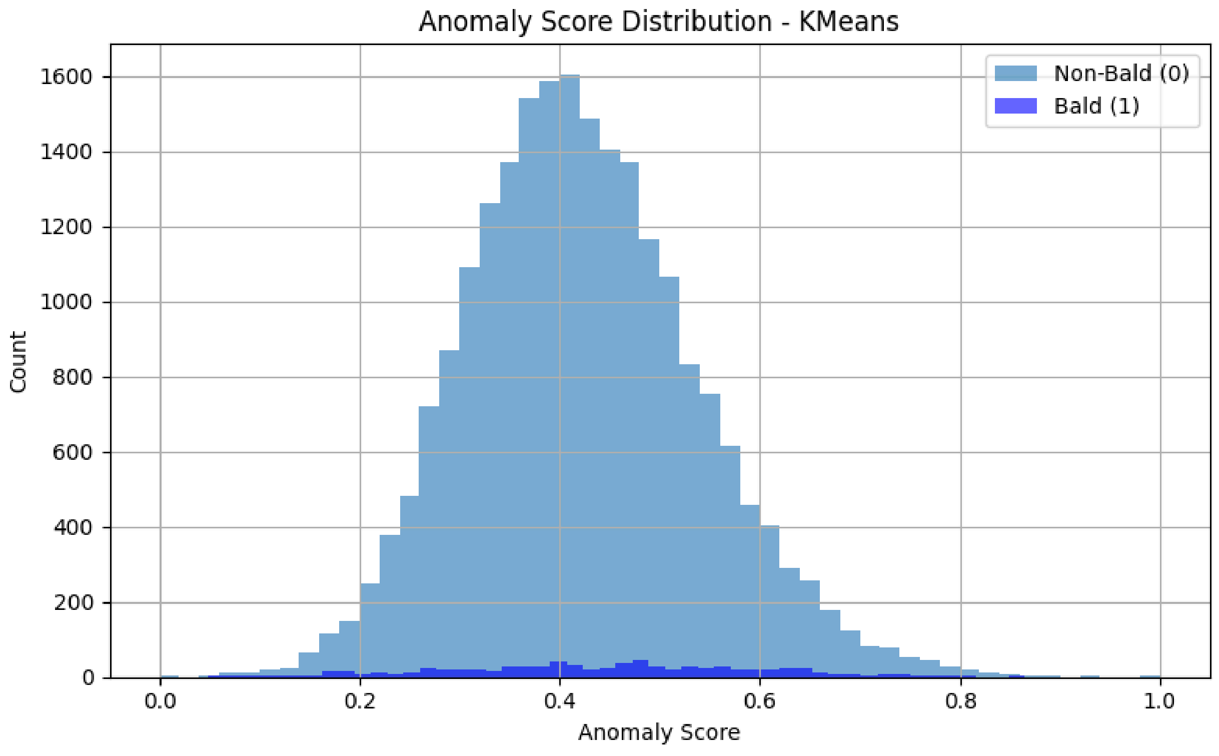
Isolation Forest distribution: Bell curve-shaped, centered around 0.4, with high overlap between classes.

9. KMeans Results

Dev Set Evaluation

AUC-ROC	Average Precision	F1 Scores
0.5552	0.0506	0.0695

KMeans shows similar performance to Isolation Forest with only slightly worse AUC and AP. Like iForest, it fails to meaningfully separate bald instances, highlighting the challenge of anomaly detection in high-dimensional, sparse binary spaces using only distance to cluster centers.



KMeans distribution: Normal curve centered around 0.4-0.5, with overlapping scores between both classes.

10. Time Evaluation

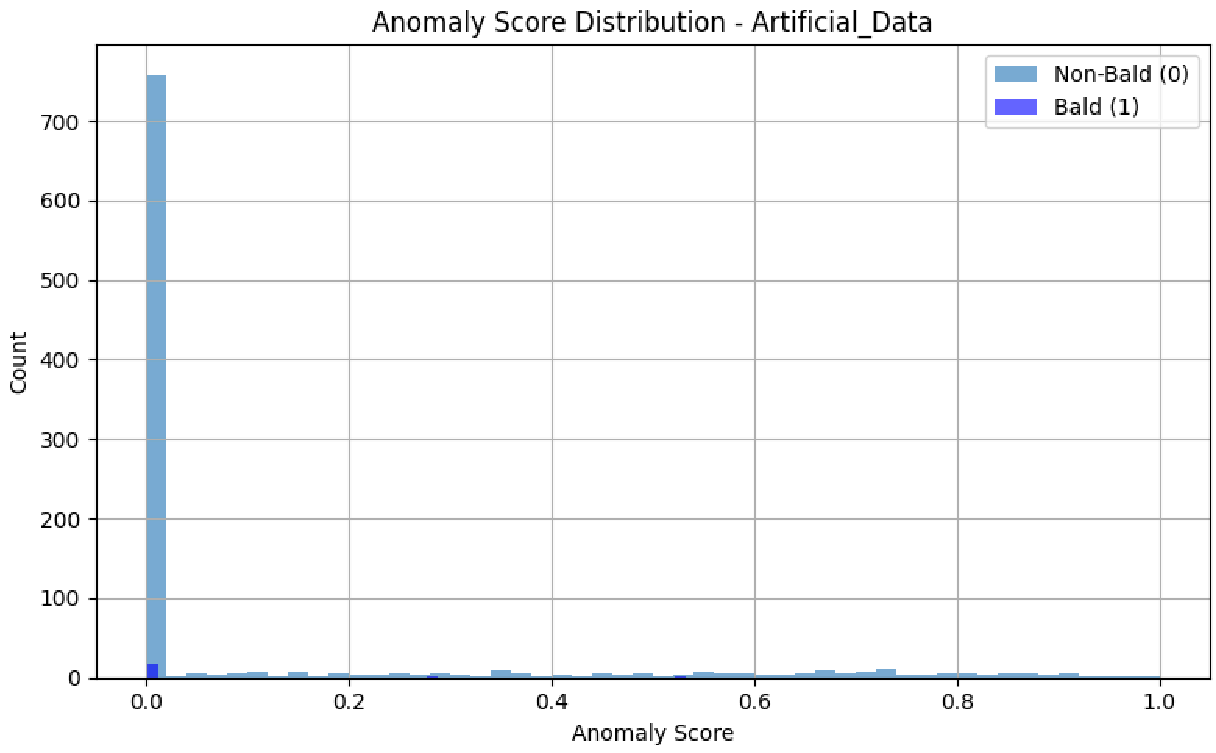
Model	Training Time (s)	Prediction Time (s)
Dummy	N/A	0.0027
DevNet	402.9888	2.0071
Isolation Forest	0.9112	0.1613
KMeans	1.3521	0.0151

Although DevNet had higher AUC, Average Precision, and F1 Scores, this also comes with a much higher training and prediction time than the other models. DevNet's training time was 4419.31% higher than Isolation Forest's and 29794.33% higher than KMean's training time. DevNet's prediction time was 1144.31% higher than Isolation Forest's and 13292.72% higher than KMean's prediction time.

10. Evaluation on Artificial Data

AUC-ROC	Average Precision	F1 Scores
0.3936	0.0161	0.0000

This result shows the challenge of applying models to synthetic data, especially when it does not reflect the underlying patterns from real-world distributions. DevNet, which performs strongly on the real dataset, fails to generalize, suggesting that its anomaly representations are sensitive to structured training data. Similarly, the unsupervised methods (iForest and KMeans) show flattened score distributions when applied to artificial input.



Artificial data scores from DevNet: No distinct clusters form, making it difficult to identify anomalies.

11. Evaluation Metrics & References

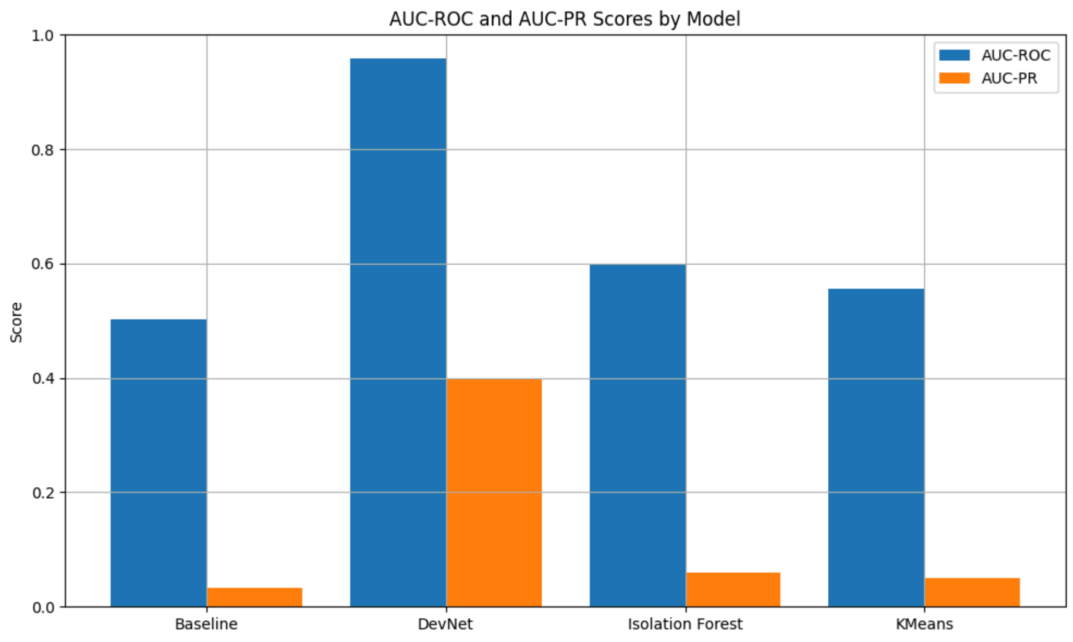
The performance of all models was evaluated using the following metrics:

AUC-ROC (Area Under the Receiver Operating Characteristic Curve): Measures how well the model ranks
anomalies above normal cases.
Average Precision (AP): Captures precision-recall tradeoff. Important in imbalanced datasets.
F1 Score: Harmonic mean of precision and recall at a specific threshold (97.8 percentile).

Time: Measures (in seconds) how long training and predictions takes for each model.

12. Conclusion

This project demonstrates the effectiveness of anomaly detection techniques for identifying rate cases, such as bald individuals in the CelebA dataset. Among the four models compared: Dummy, DevNet, Isolation Forest, and KMeans-DevNet significantly outperforms the others across AUC-ROC, Average Precision, and F1 Score, shown in the figure below. By leveraging labeled anomalies during training, DevNet is able to learn powerful representations of rare patterns, making it particularly suitable for highly imbalanced datasets.



DevNet AUC-ROC and AUC-PR outperforms alternative models.

In contrast, the unsupervised models (Isolation Forest and KMeans) showed only modest gains over random assignment. Their performance highlights the limitations of traditional methods when applied to high dimensional binary data with sparse anomalies. While Isolation Forest and KMeans require less training and prediction time, they struggle to capture meaningful distinctions between the normal and anomalous classes.

However, this improved performance with DevNet comes at a computational cost. DevNew required significantly longer training and prediction times compared to the other models, making it potentially less suitable for time constrained or resource limited applications. Results from the artificial dataset evaluation suggest that DevNet's learned anomaly representations may not generalize well to unseen distributions, underscoring the importance of distributional similarity between training and deployment data.

13. README - How to Run

1. Clone the repository:

```
git clone https://github.com/aubrie24/assignment_6.git
```

2. Create and activate a virtual environment:

```
python3 -m venv deepod-env
source deepod-env/bin/activate
```

3. Install dependencies:

```
pip install 'numpy<2.0' deepod pandas scikit-learn matplotlib joblib
```

4. Run full experiment using different models:

```
python main.py --model all
python main.py --model devnet
python main.py --model iforest
python main.py --model cluster
python main.py --model baseline
```

The following command will run all the models by default. It is used to differentiate between running on the real data and running on the artificial data. However, it does not need to be explicitly typed.

```
python main.py --type train_evaluation
```

5. Run artificial evaluation:

```
python main.py --type artificial
```

All outputs (plots, scores, saved models) are saved in the output/ directory.

References:

[1] Pang, G., Shen, C., & van den Hengel, A. (2019). Deep Anomaly Detection with Deviation Networks. In Proceedings of the 25th ACM SIGKDD Conference.
https://doi.org/10.1145/3292500.3330871
[2] Deviation Network (original implementation):
https://github.com/mala-lab/deviation-network-image
[3] DeepOD tabular DevNet framework: https://github.com/xuhongzuo/DeepOD