CSC730: Assignment 6
Measuring Performance – ROC and PR Curves

Kristophor Ray Jensen
Electrical Engineering and Computer Science
South Dakota School of Mines and Technology
Rapid City, United States
0009-0001-7344-349X

I. Introduction

In this assignment, we generate Receiver Operating Characteristic (ROC) and Precision-Recall (PR) curves using the MNIST and MNIST-C data on a probability density-based anomaly detection method of our choice. The MNIST-C dataset is a corrupted version of the original MNIST dataset, which contains images of handwritten digits. The MNIST-C dataset introduces various types of corruption to these images, such as blurring, noise, and pixelation, making it a challenging dataset for standard MNIST recognition algorithms. Our goal is to generate a composite datset of normal and anomalous images, train a model on this dataset using a label set derived from the source location, and evaluate its performance using ROC and PR curves.

II. Homework Objectives

The specific objectives of this assignment are as follows [1]:

- Get the MNIST and MNIST-C dataset.
- Select 3 different types of image corruption (e.g., canny edges) from MNIST-C.
- Create mixed datasets for training and test using the original MNIST and corrupted images at a 100/1 ratio for each corruption type.
- Create a probability density-based anomaly detector.
- Using a range of probability thresholds, create the corresponding ROC and precision-recall curves.

III. Methodology

A. Data Preparation

The first step in the assignment was to download the MNIST and MNIST-C datasets. We then selected three different types of image corruptions from the MNIST-C dataset. For each type of corruption, we created a mixed dataset for training and testing. This mixed dataset combined the original MNIST images with the corrupted images at a ratio of 100/1. This resulted in a dataset that was predominantly composed of normal images, with a small proportion of anomalous (i.e., corrupted) images.

Three mixed datasets were created using three seperate selections of corruptions. The sets of corruptions, chosen randomly, were: ”canny, dotted, stripe”, ”shear, canny, spatter”, and ”stripe, motion, scale”.

B. Visual Review

We visually reviewed the mixed datasets to ensure that the normal data, fig. 1, and anomalous images fig(s). 2 through fig(s). 8 were correctly labeled. This step was important to ensure that the anomaly detector was trained on the correct data.

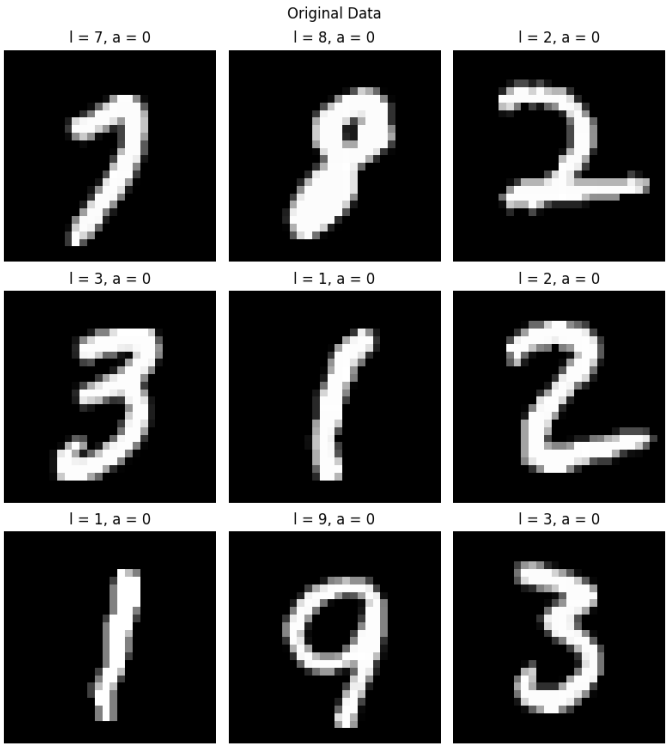

Figure 1. Original MNIST data

C. Anomaly Detection

We implemented a probability density-based anomaly detector using the implementation from scikit-learn of K-Nearest Neighbors (KNN) and Random Forest(RF). This is a supervised learning algorithm where labels of anomaly and normal data are used to train the model. The anomaly label is 1 for anomalous data and 0 for nominal data.

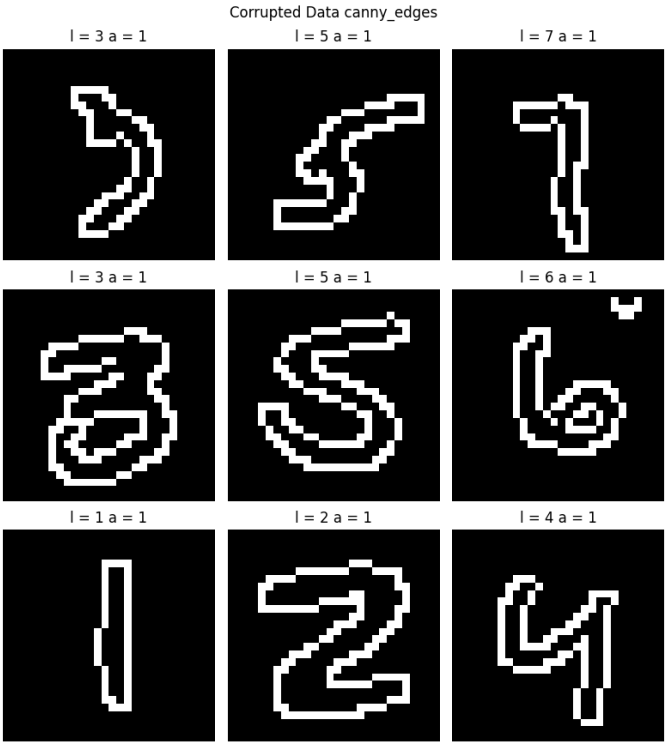

Figure 2. MNIST-C, canny edges

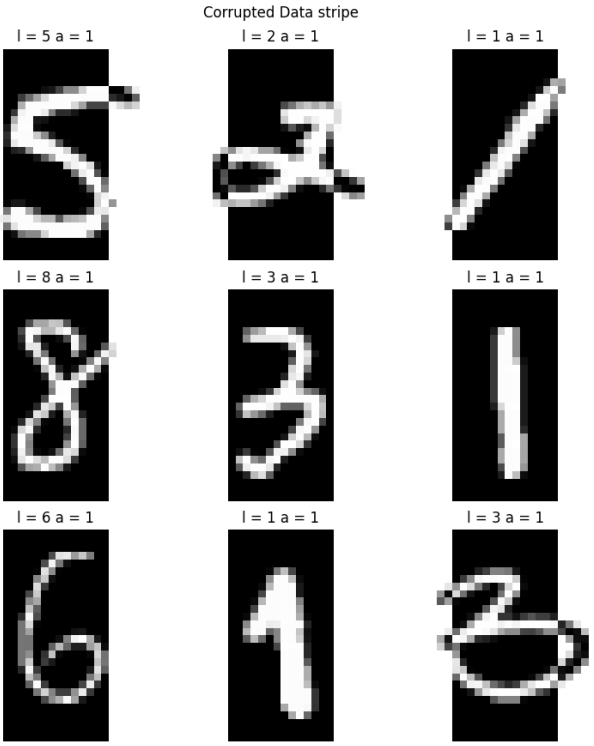

Figure 4. MNIST-C, stripe

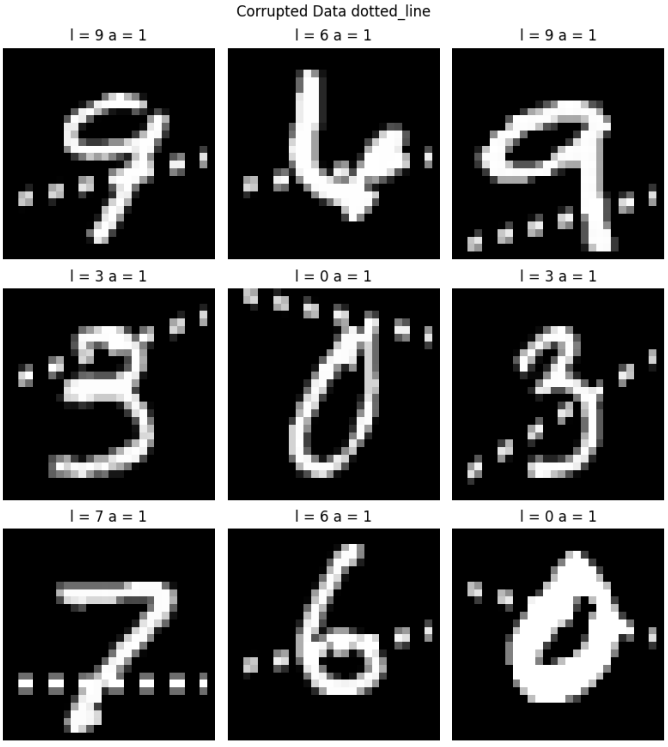

Figure 3. MNIST-C, dotted line

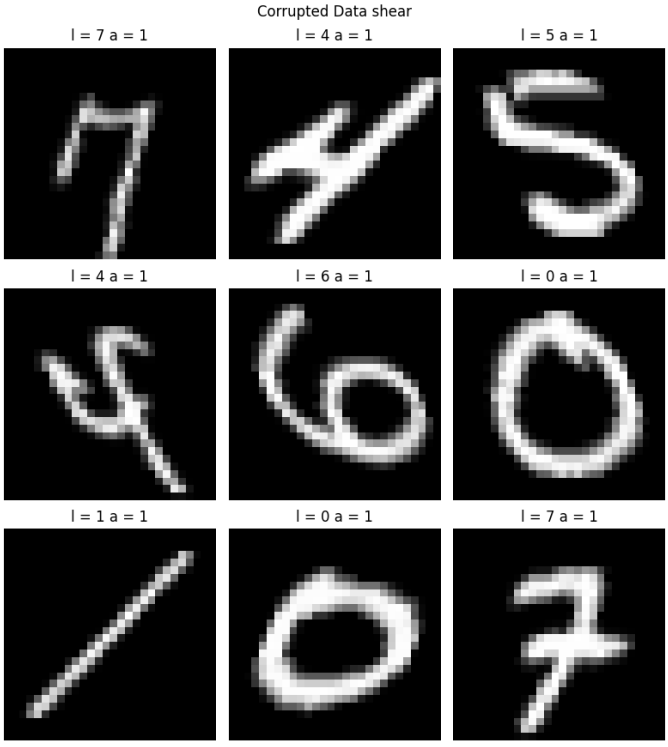

Figure 5. MNIST-C, shear

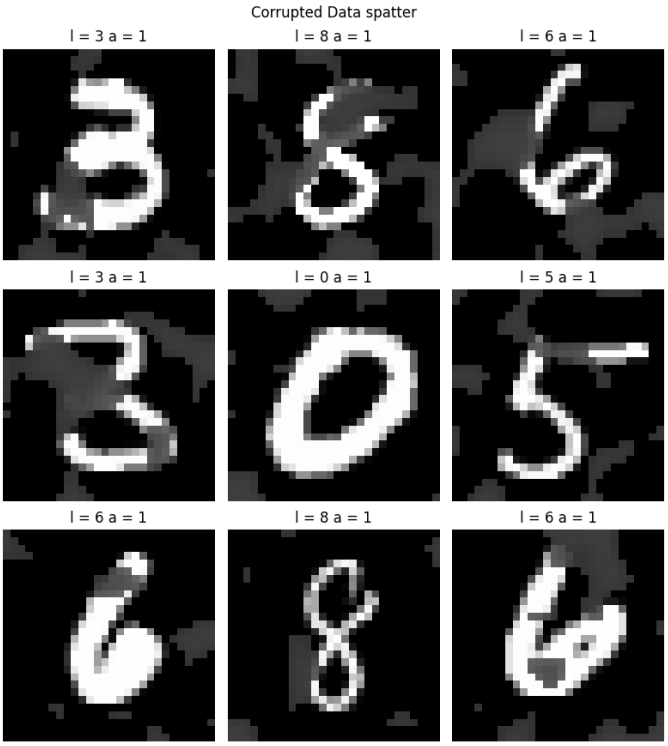

Figure 6. MNIST-C, spatter

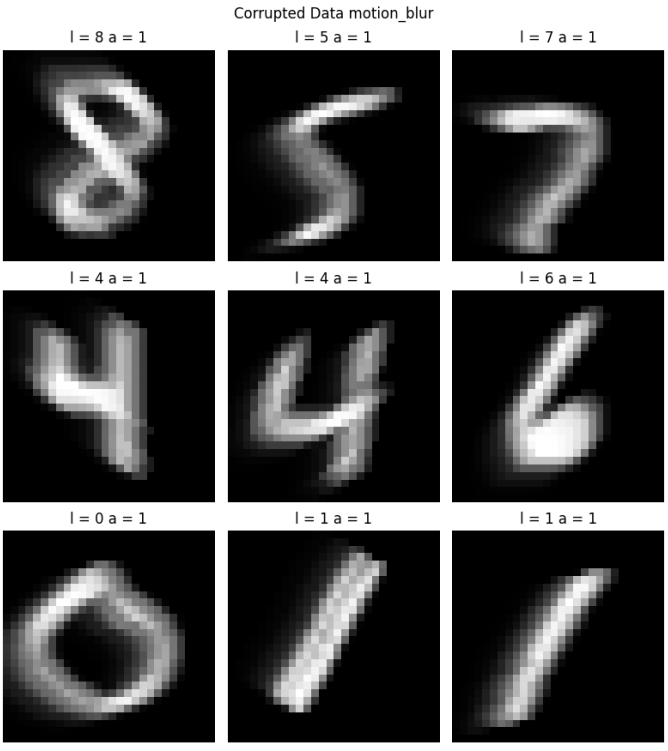

Figure 7. MNIST-C, motion blur

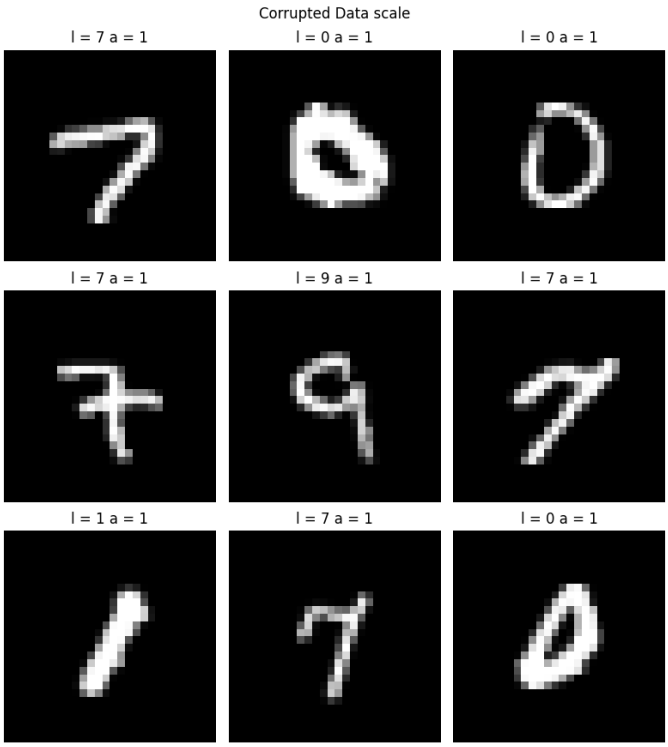

Figure 8. MNIST-C, scale

D. Performance Evaluation

We evaluated the performance of the anomaly detector using ROC and PR curves. The ROC curve plots the true positive rate against the false positive rate at different probability thresholds. A true positive is an instance where the model correctly predicts the positive class, while a false positive is an instance where the model incorrectly predicts the positive class.

The PR curve plots the precision against the recall at different probability thresholds. Precision is the ratio of true positives to the sum of true positives and false positives, while recall is the ratio of true positives to the sum of true positives and false negatives.[2]

These curves provide a visual representation of the trade-off between true positives and false positives, and between precision and recall, respectively.

IV. Results

We ran three data trials: “canny, dotted, stripe”, “shear, canny, spatter”, and “stripe, motion, scale”. For each trial, we applied the KNN and the RF classifiers from the scikit-learn library. The KNN was executed from k=3 to k=10000, and the RF was fit with estimators of 3, 10, and 30.

The KNN performance was heavily sensitive to the data selected, whereas the RF was not so sensitive. In general, the KNN had poor performance, while the RF did a very good job detecting the anomalies.

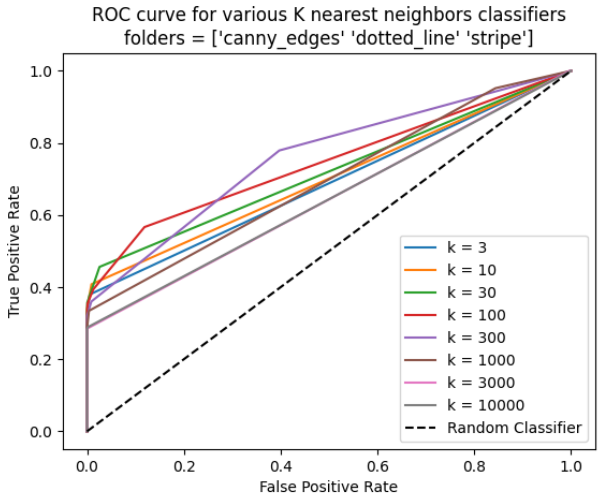

Figure 9. ROC Curve for KNN: with canny edges, dotted lines and striped

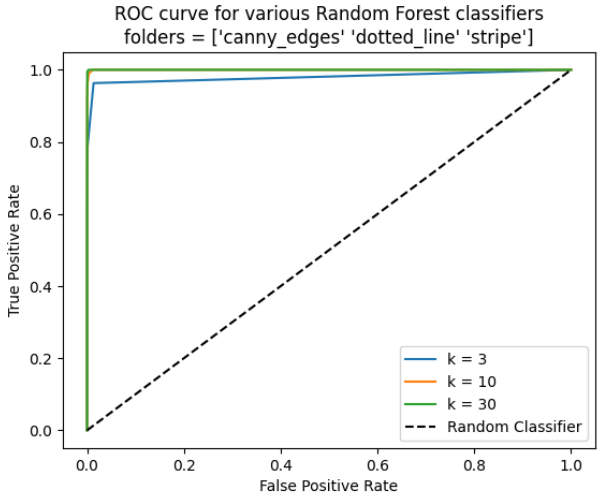

Figure 10. ROC Curve for RF: with canny edges, dotted lines and striped

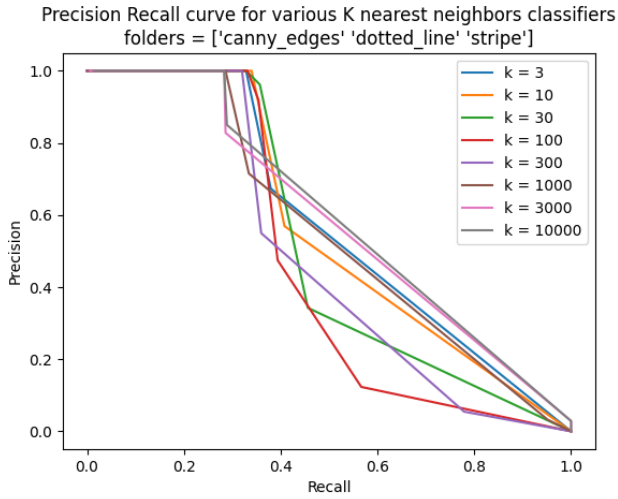

Figure 11. PR Curve for KNN: with canny edges, dotted lines and striped

Figure 12. PR Curve for RF: with canny edges, dotted lines and striped

Figure 13. ROC Curve for KNN: with shear, canny edges, and spatter

Figure 14. ROC Curve for RF: with shear, canny edges, and spatter

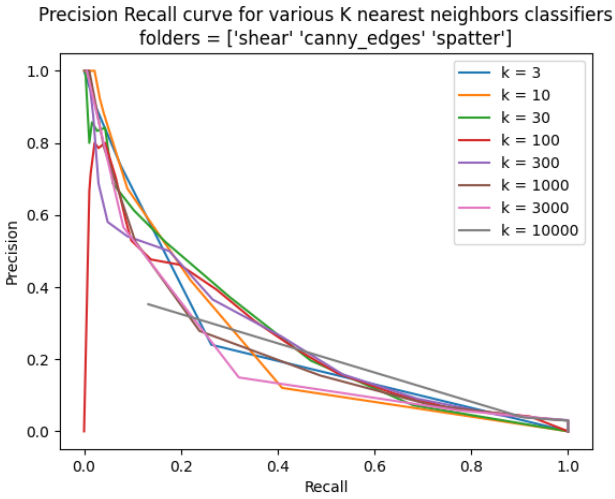

Figure 15. PR Curve for KNN: with shear, canny edges, and spatter

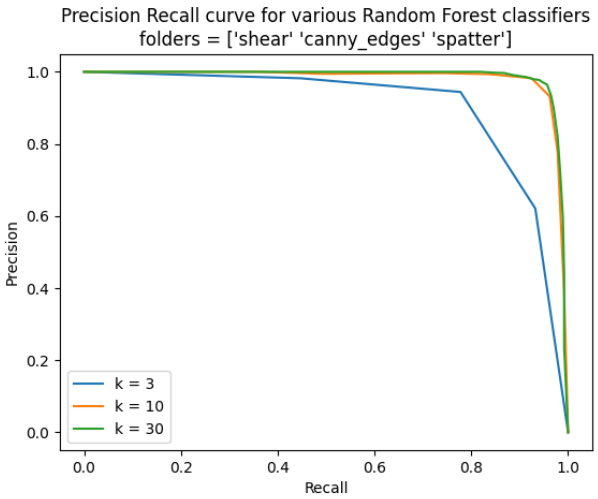

Figure 16. PR Curve for RF: with shear, canny edges, and spatter

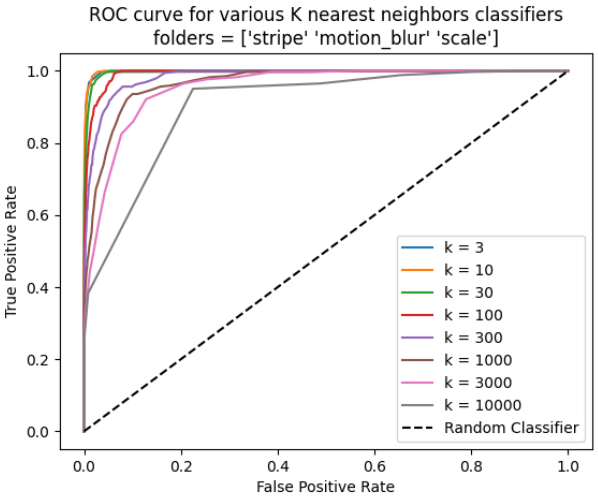

Figure 17. ROC Curve for KNN: with striped, motion blur, and scale

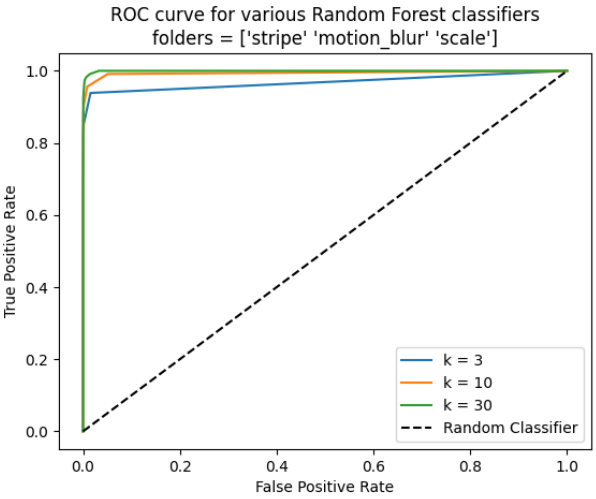

Figure 18. ROC Curve for RF: with striped, motion blur, and scale

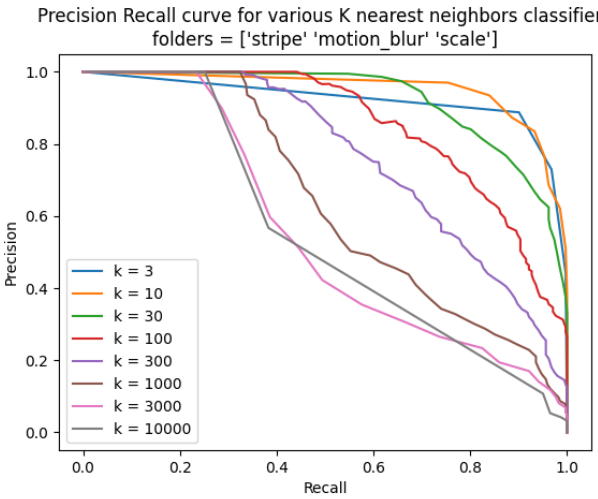

Figure 19. PR Curve for KNN: with striped, motion blur, and scale

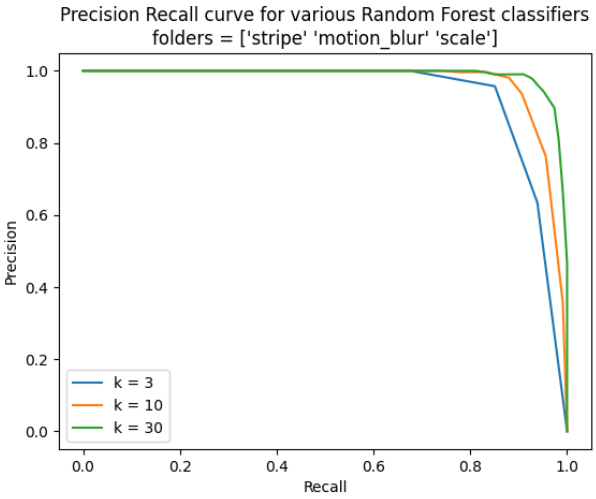

Figure 20. PR Curve for RF: with striped, motion blur, and scale

V. Conclusion

This assignment provided valuable experience to use various probability density-based anomaly detectors and evaluating their performance using ROC and PR curves. Despite the challenges posed by the MNIST-C dataset, the RF classifier performed well in detecting anomalies, while the KNN classifier struggled to achieve similar results. The RF classifier was less sensitive to the data selected, which made it a more robust choice for anomaly detection in this context.

VI. Appendix: Code Summary

The provided Python code, $csc730\_assignment\_6\_KJ.ipynb$, is a comprehensive Jupyter notebook for processing and analyzing the MNIST dataset, a large database of handwritten digits. The code also handles corrupted versions of the MNIST dataset, which are used to simulate anomalies in the data. The analysis involves the use of machine learning algorithms, specifically the k-nearest neighbors classifier (KNN) and the random forest classifier (RF). The code begins by setting up several variables related to the MNIST and corrupted MNIST datasets (MNIST-C), including paths to the data, flags indicating whether the data has been downloaded or files have been generated, and parameters for the corruption process. It also initializes empty lists to store the results of the machine learning models.

The $process\_data$ function is defined to handle the downloading and preprocessing of the MNIST data. If the data has not been downloaded, it fetches the MNIST dataset from OpenML. If the MNIST files have not been generated, it loads the data, converts it to numpy arrays, reshapes the images, and saves the processed data to disk. It also splits the data into training and testing sets.

If the $generate\_corrupted\_data$ flag is set, the code generates corrupted versions of the MNIST data. It randomly selects a number (default=3) of corruption types, applies them to a subset of the data, and concatenates the corrupted data with the original MNIST data. The labels for the corrupted data are also generated and saved. These generated labels follow this function.

$$L_a(x) = \begin{cases} 1 & \text{if } x = 1 \text{ (anomalous)} \\ 0 & \text{if } x = 0 \text{ (nominal)} \end{cases}$$

The code then loads the MNIST and corrupted MNIST data and labels from disk. It reshapes the images into a 2D format suitable for the machine learning models and splits the data into training and testing sets.

For each specified number of neighbors in $knnc\_test\_scenarios$, the code fits a KNN model to the training data, makes predictions on the testing data, and stores the predicted labels and probabilities. It does a similar process for the specified number of estimators in $rf\_test\_scenarios$ using a RF model.

The $generate\_roc\_curve$ function is defined to generate a ROC curve for the KNN and RF models. For each model and each threshold, it calculates the true positive rate (TPR) and false positive rate (FPR), and plots these rates to create the ROC curve.

The $generate\_precision\_recall\_curve$ function is defined to generate a PR curve for the KNN and RF models. For each model and each threshold, it calculates the precision and recall, and plots these values to create the PR curve.

Finally, the code calls the $generate\_roc\_curve$ and $generate\_precision\_recall\_curve$ functions to generate and save the ROC and PR curves for the KNN and RF models. The curves are saved as PNG images and displayed in the notebook.

References

[1] R. Loveland, "Assignment_6.pdf," From SDSMT D2L Website, 2024.

[2] W. Contributors, *Precision and recall*, https://en.wikipedia.org/wiki/Precision_and_recall, 20 Feb. 2024.