DATA 1030 Final Report

Kyle Lam

December 2024

1 Introduction

Anomaly detection is the characterization of small or rare events within large datasets. Automatically detecting anomalies in these datasets using data science and machine learning techniques is non-trivial as large class imbalances easily bias model predictions and produce results that discard the anomalous classes entirely. This problem stems from the idea that models can predict only the large imbalanced class, resulting in 'good' performance but unusable predictions. This also highlights the importance of using different evaluation metrics after model training as one metric, like accuracy, may not be representative of your model's performance.

In this project, we focus on an anomaly detection dataset from the UCI Machine Learning Repository named, 'Localization Data for Person Activity'. This dataset was created to develop strategies for safer smart care home environments for elderly people. This dataset contains multiple components: user IDs, sensor tags, timestamps, dates, XYZ position coordinates from sensors, and labeled activities. Previous work (primarily on Kaggle) focused on detecting anomalous events using only positional data and concatenating all of the data during training. We aim to replicate these results, but also provide an alternative approach using the calculated velocity per timestep as the primary sensor data. Through extensive preprocessing, exploratory data analysis, cross-validation, model training, and model evaluation, we aim to develop a performant machine learning pipeline to accurately tag anomalous falling events in this dataset.

2 Exploratory Data Analysis

We first begin with some feature engineering and exploratory data analysis (EDA). There are several steps we need to perform in order to create a dataset suitable for future machine learning pipelines. Using the raw table information in 1, we can define the following feature engineering steps:

Proposed Feature Engineering Steps:

1. Split the dataset into users based on user_id.

User	Sensor Tag	Timestamp	Date	X	Y	Z	Activity
A01	010-000-024-033	633790226051280329	27.05.2009 14:03:25:127	4.062931	1.892434	0.507425	walking
A01	020-000-033-111	633790226051820913	27.05.2009 14:03:25:183	4.291954	1.781140	1.344495	walking
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 1: Raw dataset imported into a pandas DataFrame with column labels. The first two rows are shown as an example of what the raw dataset looks like.

User	Sensor Tag	X	Y	Z	Activity	Anomaly
A01	010-000-024-033	4.062931	1.892434	0.507425	walking	0
A01	020-000-033-111	4.291954	1.781140	1.344495	walking	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 2: Truncated and cleaned dataset using the data from Table 1 as the basis. The first two rows are again shown to indicate how the dataset now looks.

2. Add a binary 'anomaly' column that indicates whether the activity is anomalous ('falling') or not (any other activity).

3. Remove columns that do not make sense in the context of anomaly detection.

Steps 1-2 are mainly for feature engineering and data cleaning and do not alter the results of future EDA steps. Step 3 involves the removal of the 'Timestamp' and 'Date' columns, as the detection of falling events could happen at any time of the day and should not rely on a unique timestamp identifier. With this, we are left with columns containing information for the user, sensor tag, XYZ positions, activity, and anomaly, as described in Table 2. These steps are described in the original Kaggle dataset that this project is based on.

3 Methods

Since my dataset is user-specific, a time series dataset, and a heavily imbalanced dataset, there are several splitting factors in play. We need to do the following when splitting the data:

1. Ensure the data is split on user IDs and the data from one user is only found in the train/val/test set.

2. Ensure we split correctly on the time-series to avoid data leakage.

Algorithm	Parameters	Optimal Parameters
LogisticRegression	C = [0.1, 1, 10]	C = 0.1
RandomForest	max_features = [0.25, 0.5, 1.0] max_depth = [5, 10, 30]	max_depth = 30 max_features = 1.0
KNearestClassifier	n_neighbors = [1, 5, 10, 30] weights = ['uniform', 'distance']	n_neighbors = 5 weights = 'distance'
XGBoostClassifier	n_estimators = [100, 500, 1000, 2000] max_depth = [3, 4, 5]	n_estimators = 2000 max_depth = 3

Table 3: Selected models and training parameters for a grid search cross-validation training. The optimal parameters for each model after performing a grid search are listed.

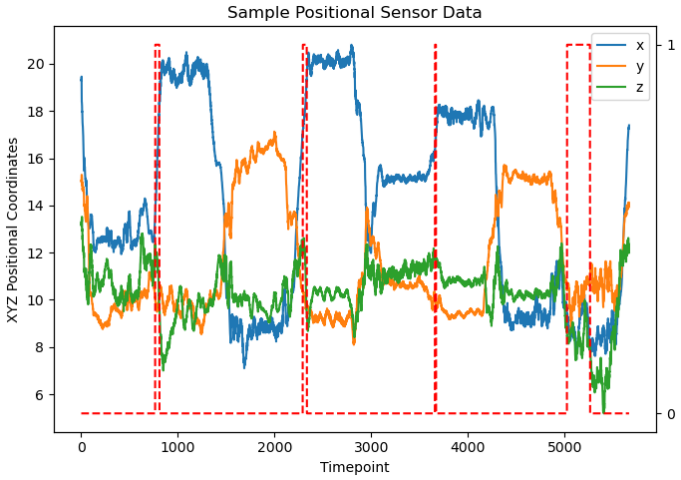

Figure 1: Sample XYZ positional data from User A01 with anomalous regions covered in red dashed lines.

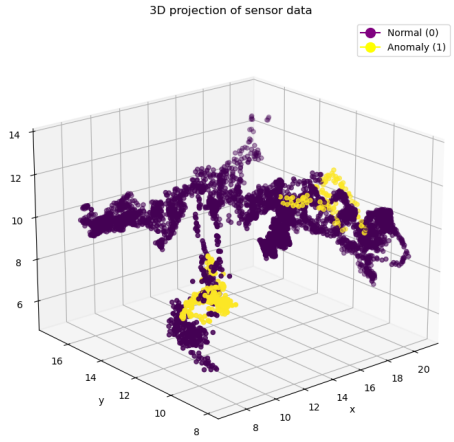

Figure 2: 3D projection of XYZ positional coordinates with anomalous events highlighted in yellow.

We can address these splitting issues by using the GroupShuffleSplit and GroupKFold methods in the sklearn package. Using GroupShuffleSplit, we divided the dataset into training and testing sets, which ensures no overlap of user data between the two sets. This prevented data leakage and avoided overfitting to specific users. Within the training set, we used GroupKFold cross-validation with three folds. Group-based splitting ensured that the same user's data did not appear in either the training and validation sets within each fold, maintaining independence and allowing for accurate model evaluation.

The data preprocessing pipeline was handled using a ColumnTransformer to accommodate different feature types. Categorical features, like the sensor tags, were one-hot encoded to represent them as binary vectors, while numerical features like X, Y, and Z were standardized using a StandardScaler to normalize their distributions. The User label was excluded from the model input but used in the above-mentioned grouping to ensure that model predictions were independent of user identity. This process follows the ML pipeline search described in PS7 and adapts several components from the lecture notes.

We used recall as the primary evaluation metric to measure model performance. Recall was chosen because it emphasizes minimizing false negatives, which is critical for this dataset involving fall detection. Misclassifications, such as identifying "falling" as any other "normal" activity, could lead to misleading conclusions and more significant consequences than occasional false positives.

To evaluate the models, we tested four machine learning algorithms, including both linear and non-linear models. Logistic regression with L2 regularization served as a baseline linear model. For non-linear models, we used Random Forest, XGBoost, and K-Nearest Neighbors. The tested parameters and optimal parameters found are in Table 3.

Algorithm	Mean Recall Score	Standard Deviation Recall Score
Baseline	0	0
LogisticRegression	0.0109	0.0216
RandomForest	0.1217	0.0351
KNearestClassifier	0.1624	0.0610
XGBoostClassifier	0.1778	0.1114

Table 4: Results for the initial GridSearchCV pipeline using recall as the main scoring function. The baseline recall score of the anomalous column is 0 since no positive class values will be predicted.

4 Results

Overall, the models initially appear to perform poorly when evaluating performance. XGBoost has the best performance, followed by KNearestClassifier, RandomForest, and finally Logistic Regression.

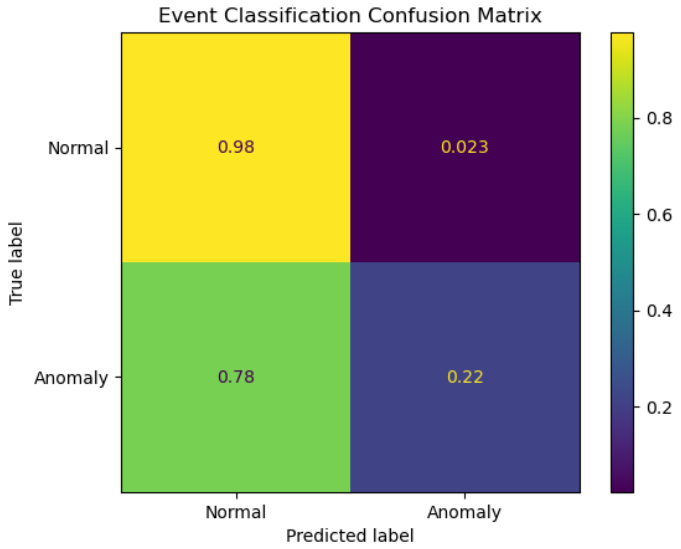

Figure 3: Confusion matrix of anomalous events using the best-performing hyperparameters of the XGBoostClassifier model.

When we analyze the confusion matrix in Figure 3 using the best performing model, XGBoost, we can notice several expected and unexpected results. We expected to see a high non-anomalous true negative score due to the imbalanced dataset. Given our initial recall score performance, we expect to see poor performance on the true positives for anomaly events. The unexpected result is the relatively high false negative score, which has a value of 0.78. As discussed earlier in the Methods, a high false negative score is disastrous for this type of

fall detection model, as it means that we fail to identify the majority of 'falling' events.

While the initial results appear to be bad, we have to further explore the predictions. When we plot the predicted anomalous points onto a test set in Figure 4, we can see how the evaluation metrics fail to show the full picture.

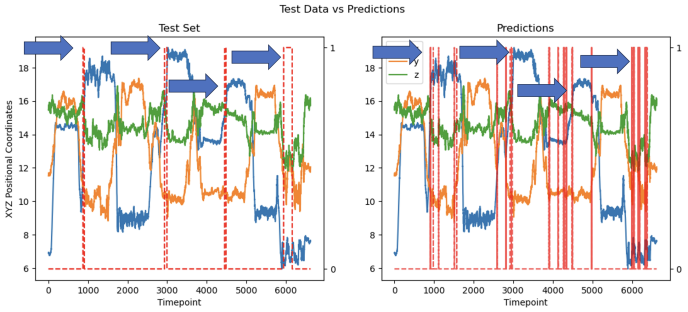

Figure 4: (Left) Test set with XYZ positional coordinates and the start of anomalous regions selected. (Right) The same test set with XYZ positional coordinates and predict anomalous zones from the best-performing XGBoost-Classifier model.

While the model predicts various anomalous regions, we can see that the model does accurately predict the start of the same anomalous regions found in the test set. The issue with the large false negative value is a result of the model failing to predict a continuous anomalous zone. The model produces discrete predictions, and not all points in an anomalous zone may be contained in the final prediction. As long as the model accurately predicts the start of an anomalous zone, we can use this as a preliminary alert for caretakers utilizing this system for automated supervision.

When we look at the SHAP global scores in Figure 5, we see that the XYZ position values are the most important features, with the sensor locations playing a minimal role in model predictions. The Y position appears to be the most important feature on this SHAP value graph, as the values are more clearly defined as either low or high importance to model predictions. This makes sense in the context of the dataset, as the Y axis is typically associated with vertical position and movement. A sudden drop in the Y position likely indicates a 'falling' event, while stationary Y values indicate other 'normal' events like walking or sitting.

5 Outlook and Limitations

While I was able to successfully train the model based on the raw data downloaded from Kaggle, a discussion following my project presentation highlighted several issues with my implementation:

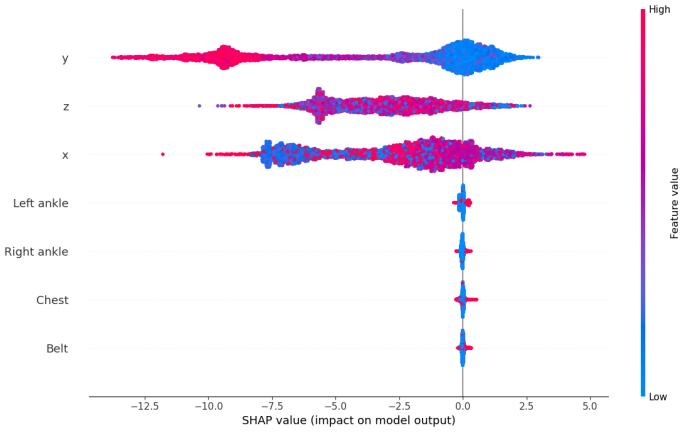

Figure 5: SHAP values for model predictions on the XGBoostClassifier.

1. Using XYZ positional data may be the wrong approach for this project, as position data is relative to the user's environment and may not be representative of any specific action.
   (a) Fix: use velocity or acceleration data by calculating it manually.
2. Interpretability on the XYZ positional data does not yield meaningful interpretations using the same reasoning as in 1.

My preliminary results showed some meaningful predictions on test datasets (as shown in the results), but the quality and integrity of the dataset are highly questionable.

After presenting, I searched a bit deeper into the original dataset from the UCI Machine Learning repository that the Kaggle one was based off on. When using this dataset and performing a second EDA process, I noticed that there were several inconsistencies between the two datasets:

1. XYZ position data from Kaggle did not align with the XYZ data from the UCI ML repo (Figure 1 vs Figure 6).
   (a) In the UCI ML dataset, the XYZ data appears to come from an accelerometer instead of positional sensor data.
   (b) In the UCI ML dataset, the data points rarely reach above a value of 6 while the Kaggle dataset consistently reaches values of 16+.
   (c) In the UCI ML dataset, some data points reach the negatives, while the Kaggle dataset does not contain any negative values.
2. Activity and anomalous 'falling' counts did not align between the two datasets.

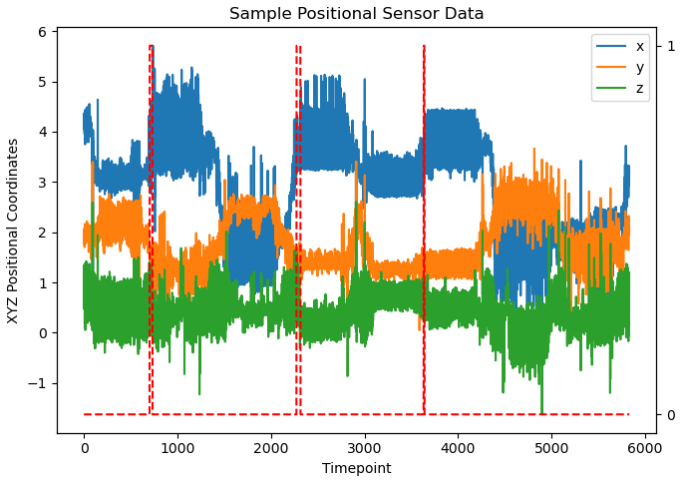

Figure 6: XYZ sensor data from User A01 from the UCI Machine Learning Repository. Note that these XYZ values on this graph compared to that of Figure 1.

(a) There are more total anomalies and anomalous regions in the Kaggle dataset versus the UCI ML dataset. 8183 anomalous events in the Kaggle dataset versus 2973 anomalous events in the Kaggle dataset.

Unfortunately, the creator of the Kaggle dataset included minimal data cleaning steps when preparing their dataset, so it is extremely difficult to replicate. The only thing that is linked is the original dataset from the UCI ML repository.

This poses a significant issue when evaluating the model performance and dataset quality. These findings prompted me to conduct an expanded study using the UCI ML dataset in order to reevaluate model performance. While I did attempt to revise my project and models, I unfortunately did not have enough time to train and troubleshoot within the scope and time left for this report. Even relying when relying Oscar to run multiple iterations of the notebook, the relatively large size of the dataset (approx. 160,000 rows) made it difficult to troubleshoot within a reasonable time. Therefore, the majority of this report displays my old results on the Kaggle dataset, instead of the original dataset from the UCI ML repository.

6 References

[1] Vidulin, V., Lustrek, M., Kaluza, B., Piltaver, R., Krivec, J. (2010). Localization Data for Person Activity [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C57G8X.