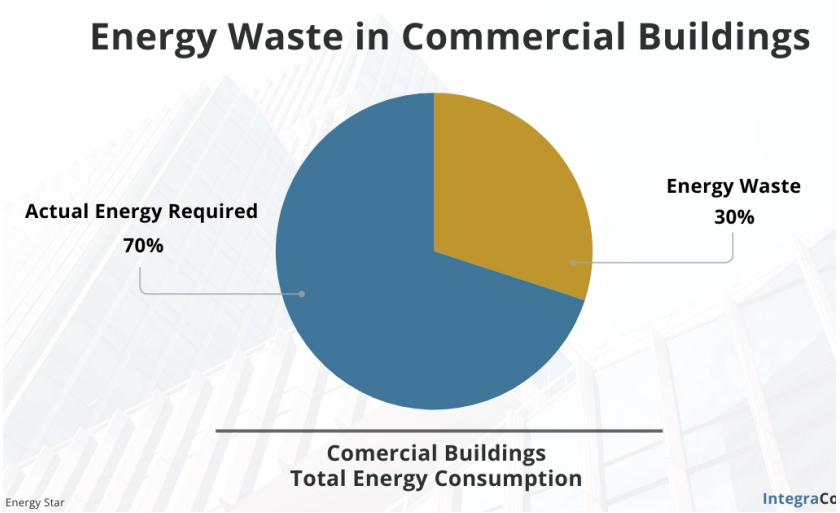
Anomaly Detection in Energy Consumption

Anushka Arun

December 17, 2025

Motivation

- Buildings consume approximately **one-third** of the total energy around the world.
- It is estimated that **more than 20%** of this energy is wasted due to equipment failure, aging, misconfiguration, and human-related errors.
- This wasteful use of energy can be identified and prevented using a data-driven analytical technique known as anomaly or outlier detection.
- Data from energy sensors can be leveraged to identify anomalies present in the energy consumption profiles, which is a big step towards saving energy and achieving global sustainability



Background

- Energy sensors that keep track of energy consumption provide minute, consistent data all year-round
- Current ML models often rely on algorithms that require offline training on “clean” or labelled data, which is costly and labour-intensive
- Difficult to obtain supervised energy consumption data
- Best to employ anomaly detection methods that don’t require much supervision – unsupervised methods and semi-supervised methods
- Making use of NN models is also useful for time-series prediction, and can then be used for
  - Understanding energy consumption trends
  - Optimizing energy usage
  - Forecasting electricity demand

Related Work

Past research has tried a variety of different models for unsupervised anomaly detection. A few relevant results to note from various studies:

- Existing methods like RPCA and SAX suffer from one or more of the following limitations
  - Lack of robustness against noise
  - Inability to perform online anomaly detection
  - Non-linear computational complexity as more data is observed
- LSTM models are particularly good for time series
  - Robust to Noisy Data
  - Capturing Non-Linear Patterns
  - Long-Term Dependency Learning

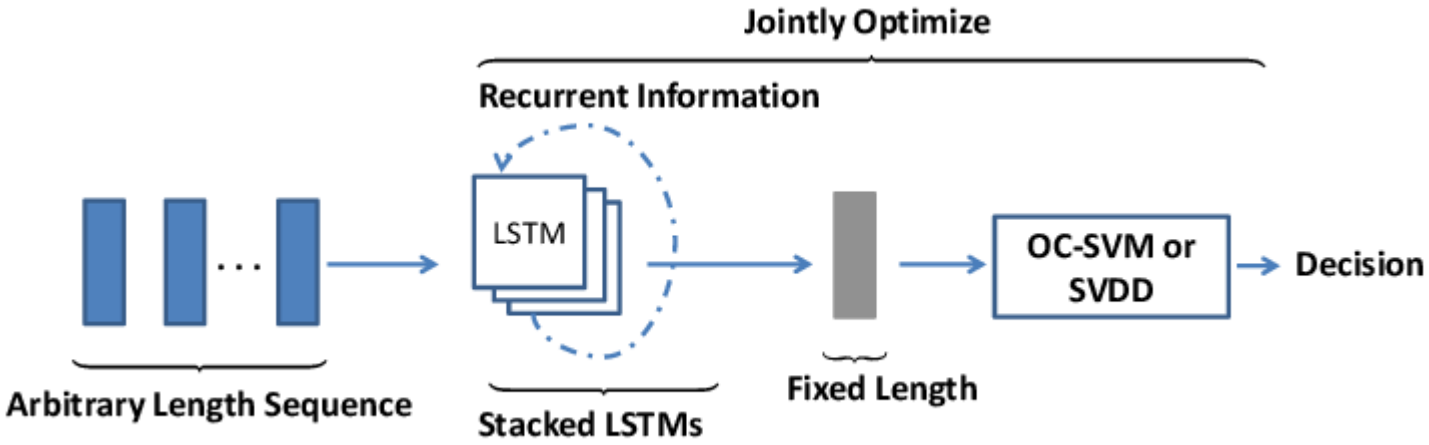
Claim / Target Task

Is it possible to detect anomalies relatively soon after they occur using unsupervised or semi-supervised LSTM hybrid models?

More specifically, can we obtain and compare the precision, recall, and F1 score of the semi-supervised LSTM-OC-SVM, unsupervised LSTM-KNN, unsupervised LSTM-DBSCAN hybrid models?

- Do semi-supervised methods perform better than unsupervised methods?

Proposed Solution



- LSTM Autoencoder is designed to reconstruct its input sequence
- An **encoder** LSTM that compresses the input into a low-dimensional latent representation
- A **decoder** LSTM that attempts to reconstruct the original sequence from this compressed code
- Compresses each input sequence into a **latent vector**, a compact representation that captures the essential temporal patterns of the data
- Latent vectors form a new feature space where normal and abnormal behaviors may become **more separable**

Proposed Solution

- Instead of manually setting a fixed threshold for the reconstruction error, we apply **clustering** to group similar latent representations.
- **Normal sequences cluster together**, while **anomalous ones appear as outliers** or form smaller, distinct clusters.

Clustering methods

- OC-SVM
  - Learns a boundary around a set of "normal" data points to identify deviations
- KMeans
  - Iteratively partitioning a dataset into K clusters
  - Identifies outliers, data points that lie significantly far from any cluster centroid, which are considered anomalies because they do not conform to the established "normal" clusters of data patterns
- DBSCAN
  - automatically classifies points with high reconstruction errors (which form low-density regions in the error space) as noise points, effectively identifying them as anomalies

Implementation

```
def LSTM_compression(data):

  input_dim = data.shape[2]
  timesteps = data.shape[1]

  inputs = Input(shape=(timesteps, input_dim))
  encoded = LSTM(64, activation='relu', return_sequences=False, name="encoder")(inputs)
  decoded = RepeatVector(timesteps)(encoded)
  decoded = LSTM(64, activation='relu', return_sequences=True)(decoded)

  autoencoder = Model(inputs, decoded)
  autoencoder.compile(optimizer='adam', loss='mse')

  autoencoder.fit(data, data, epochs=30, batch_size=64, validation_split=0.1, shuffle=False)

  encoder_model = Model(inputs, encoded)
  latent_vectors = encoder_model.predict(data, verbose=1, batch_size=32)  # shape = (num_samples, 64)

  return encoder_model, latent_vectors
```

Keras LSTM model used for compression and reconstruction of inputs into latent vectors

Implementation

```python
def hyperparameter_tuning():
    best_score = 0
    best_params = {}
    precisions = []

    for nu_val in [0.1, 0.15, 0.2]:
        for gamma_val in ['auto', 'scale']:
            normal_train_vectors = train_latent_vectors[Y_train.iloc[:, 0] == 0]

            oc_svm = OneClassSVM(nu=nu_val, kernel='rbf', gamma=gamma_val)
            oc_svm.fit(normal_train_vectors)

            # Evaluate on the validation data
            # Predictions will be +1 (inlier/normal) or -1 (outlier/anomaly)
            val_latent_vectors = encoder_model.predict(X_val, verbose=0, batch_size = 32)
            val_predictions = oc_svm.predict(val_latent_vectors)

            # Remap predictions to 0 (normal) and 1 (anomaly) for comparison with y_test_labels
            remapped_predictions = np.zeros_like(val_predictions)
            remapped_predictions[val_predictions == -1] = 1

            # Calculate metrics
            precision = precision_score(Y_val, remapped_predictions, pos_label=1)
            precisions.append(precision)
            recall = recall_score(Y_val, remapped_predictions, pos_label=1)
            f1 = f1_score(Y_val, remapped_predictions, pos_label=1)

            print(f"Params: nu={nu_val}, gamma={gamma_val} | Precision: {precision:.4f}, Recall: {recall:.4f}, F1-Score: {f1:.4f}")

            # Keep track of the model with the best Precision (TP/FP + TP)
            if precision > best_score:
                best_score = precision
                best_params = {'nu': nu_val, 'gamma': gamma_val}

    print(f"\nBest parameters found: {best_params} with Precision score of {best_score:.4f}")
    return best_params, best_score, precisions
```

Trained OC-SVM on labelled training data to represent a semi-supervised method, and used hyperparameter tuning to find the best OC-SVM model before fitting on testing latent vectors

Implementation

```
# Apply KMeans clustering to group similar latent representations
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, random_state=42)
labels = kmeans.fit_predict(test_latent_vectors)

# We assume the larger cluster is "normal"
normal_cluster = np.bincount(labels).argmax()
anomaly_mask = labels != normal_cluster
```

```
# Convert boolean mask to integer labels (0 for normal, 1 for anomaly)
kmeans_anomaly_labels = np.zeros_like(anomaly_mask, dtype=int)
kmeans_anomaly_labels[anomaly_mask] = 1
```

Applied Kmeans clustering to LSTM Autoencoder generated latent representation vectors

Implementation

```
# Clustering algorithm that identifies data points in sparse regions as noise or anomalies

from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.5, min_samples=10)
clusters = dbscan.fit_predict(latent_pca)

# Initialize all points as 'normal' (class 0)
remapped_labels = np.zeros_like(clusters)
remapped_labels[clusters == -1] = 1
```

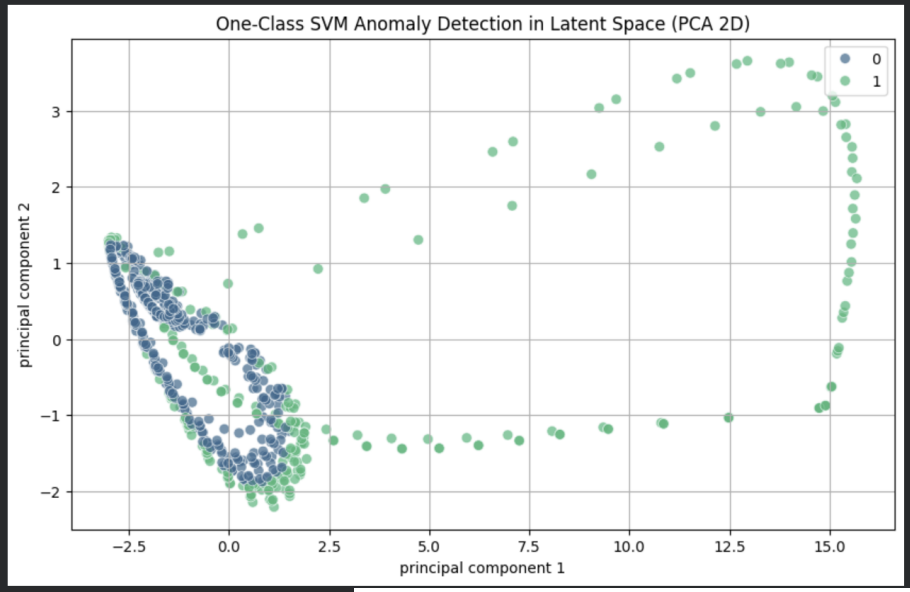
DBSCAN was also implemented as another clustering model used to identify anomalies from the latent representation

Implementation

```
precision = precision_score(Y_test, anomaly_labels, pos_label=1)
recall = recall_score(Y_test, anomaly_labels, pos_label=1)
f1 = f1_score(Y_test, anomaly_labels, pos_label=1)

print(f"Precision: {precision:.4f}, Recall: {recall:.4f}, F1-Score: {f1:.4f}")
```

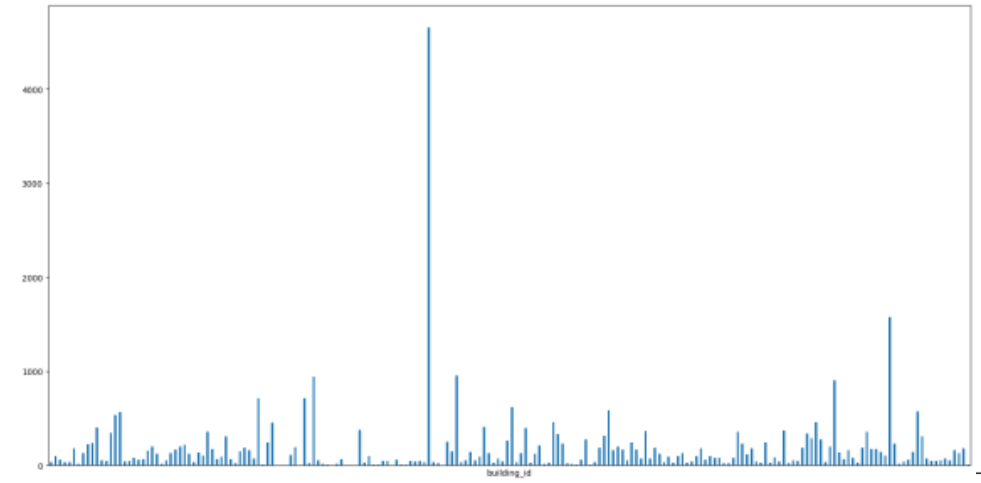
```
Precision: 0.2215, Recall: 0.8148, F1-Score: 0.3483
```

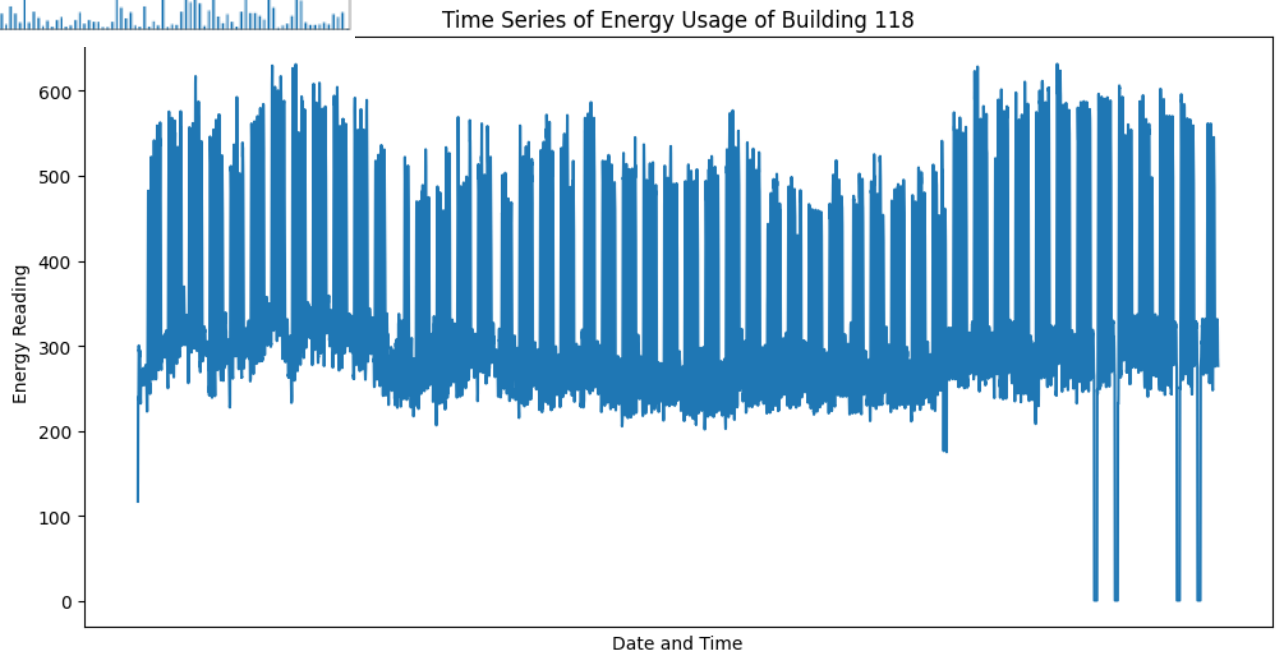
```
# Visualize the latent space with PCA

from sklearn.decomposition import PCA

pca = PCA(n_components=2)
latent_pca = pca.fit_transform(test_latent_vectors)

plt.figure(figsize=(10, 6))
sns.scatterplot(x=latent_pca[:, 0], y=latent_pca[:, 1], hue=anomaly_labels, palette='viridis', s=50, alpha=0.7)
plt.title("One-Class SVM Anomaly Detection in Latent Space (PCA 2D)")
plt.xlabel("principal component 1")
plt.ylabel("principal component 2")
plt.grid(True)
plt.show()
```



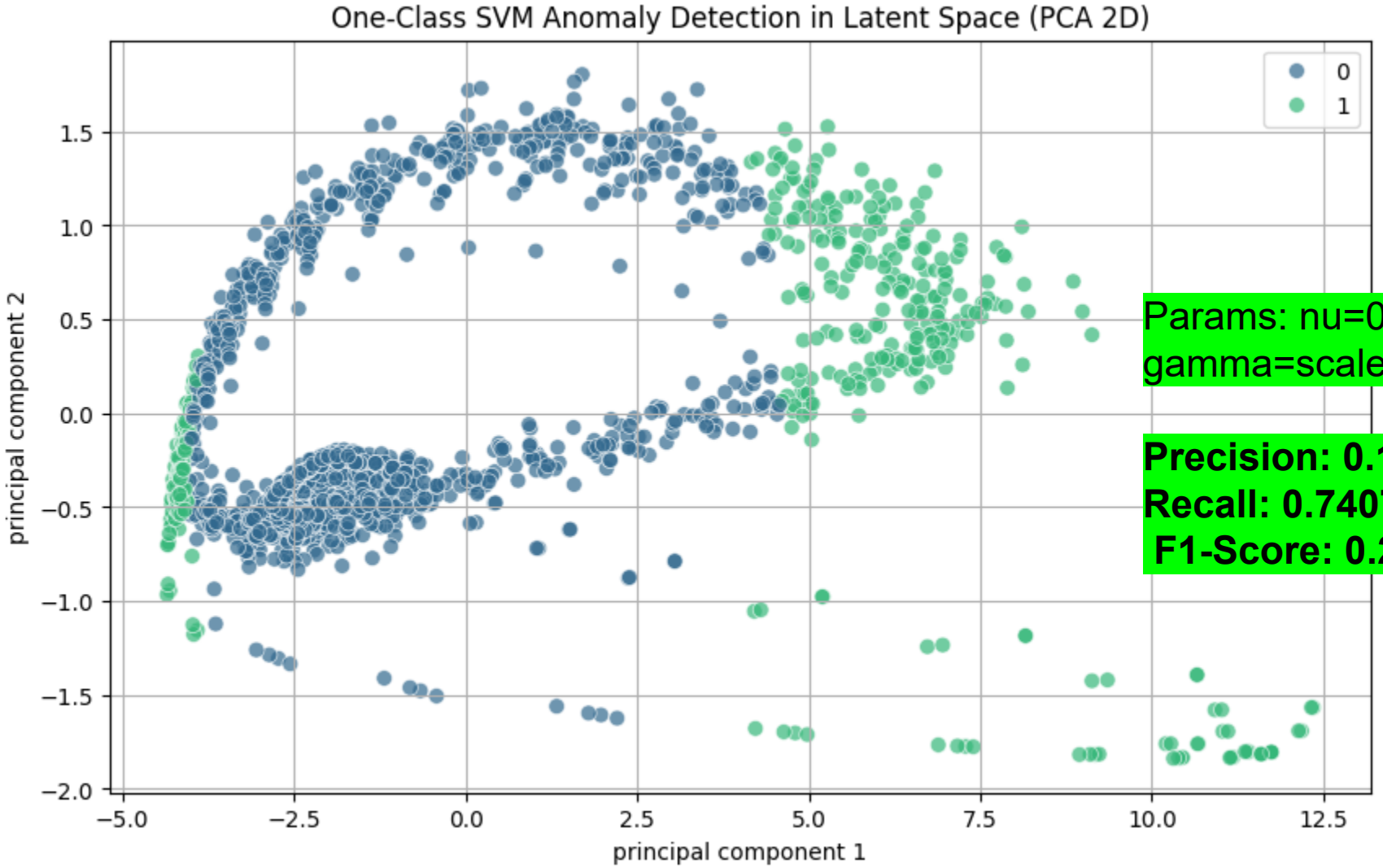
Utilized PCA to visualize the latent space, and precision, recall, and f1 score as metrics to comparatively analyze the models

Data Summary





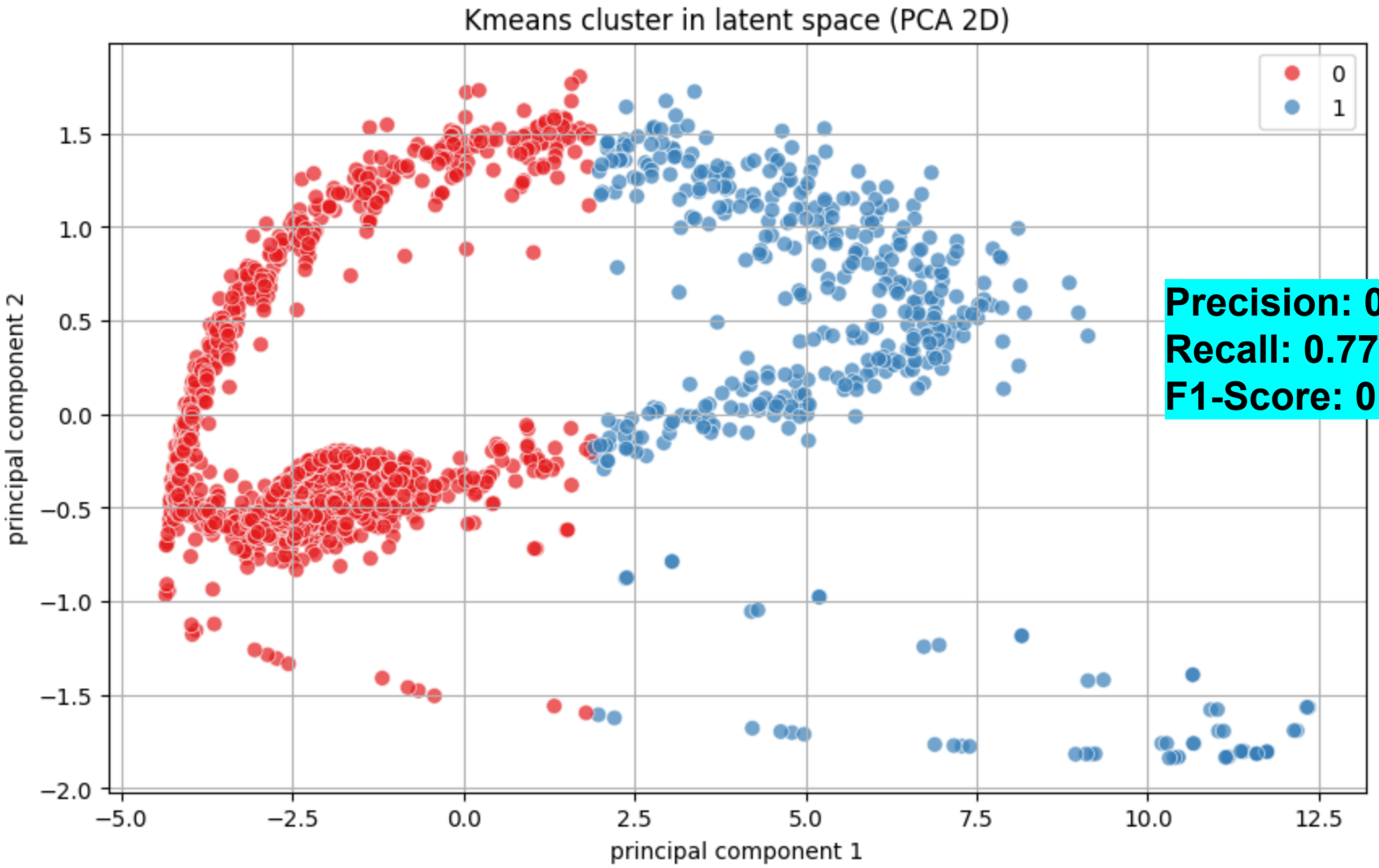
Experimental Results



Params: nu=0.2, gamma=scale |

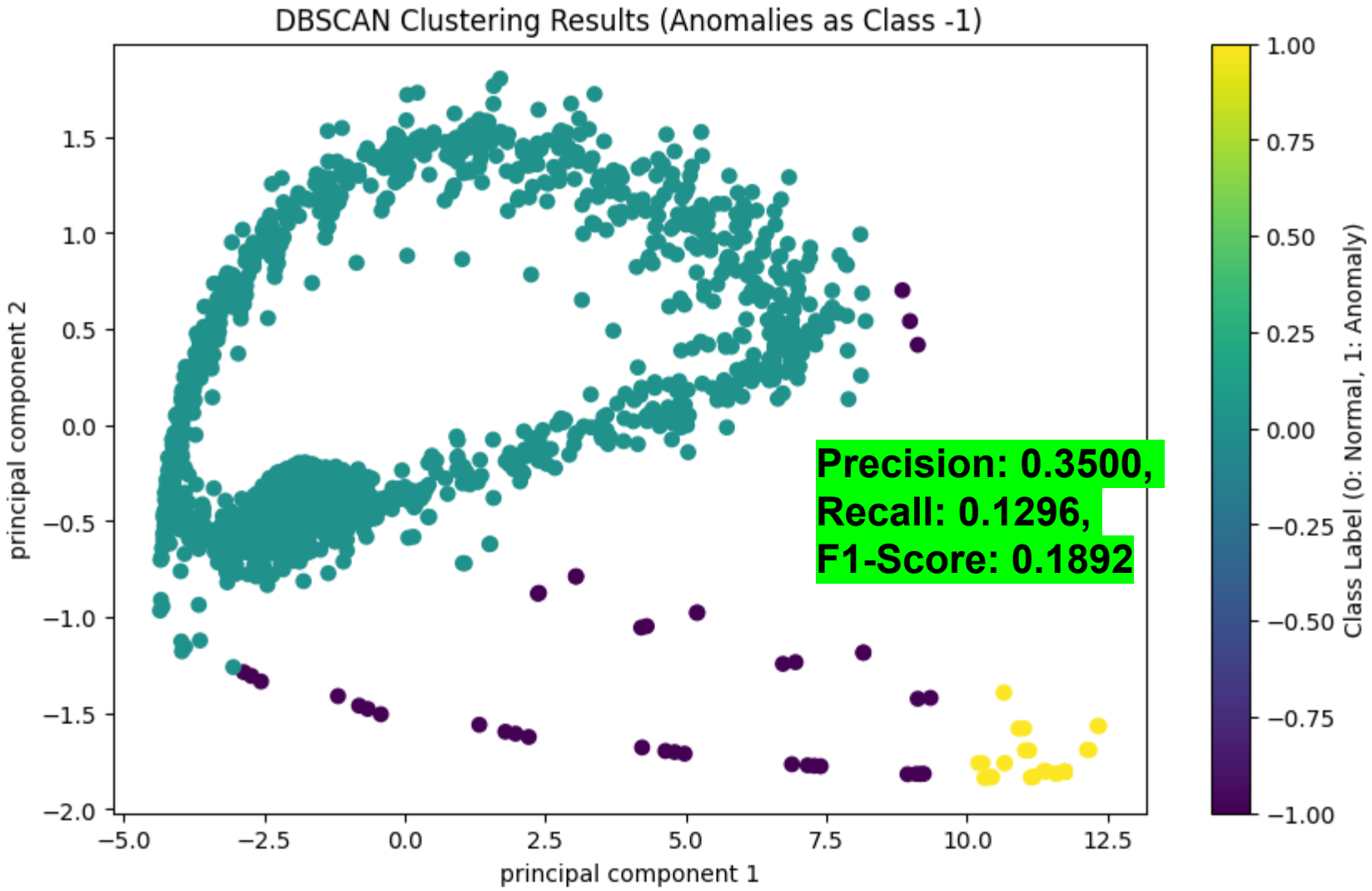
Precision: 0.1550, Recall: 0.7407, F1-Score: 0.2564

Experimental Results

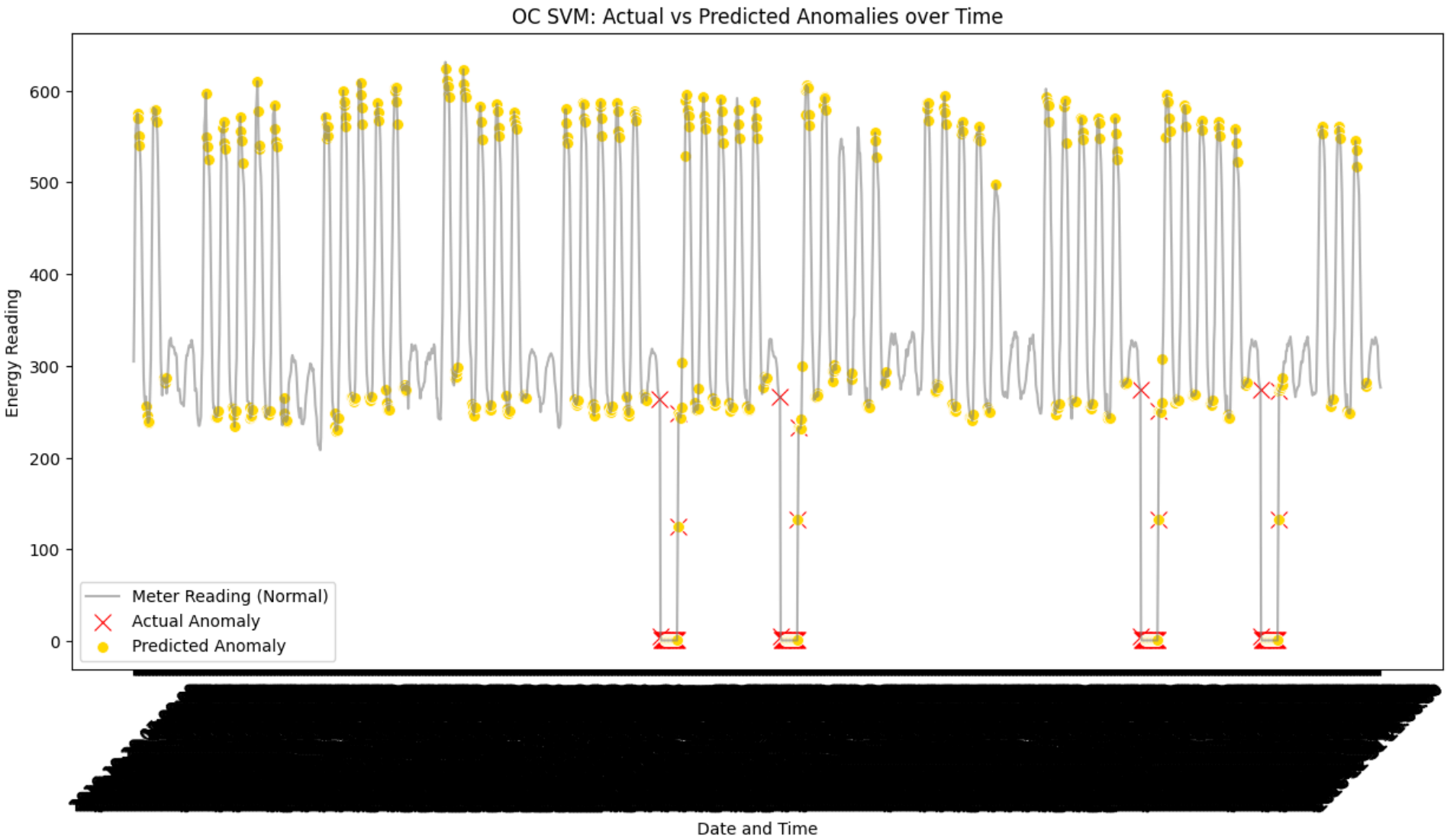


Precision: 0.1776,
Recall: 0.7778,
F1-Score: 0.2892

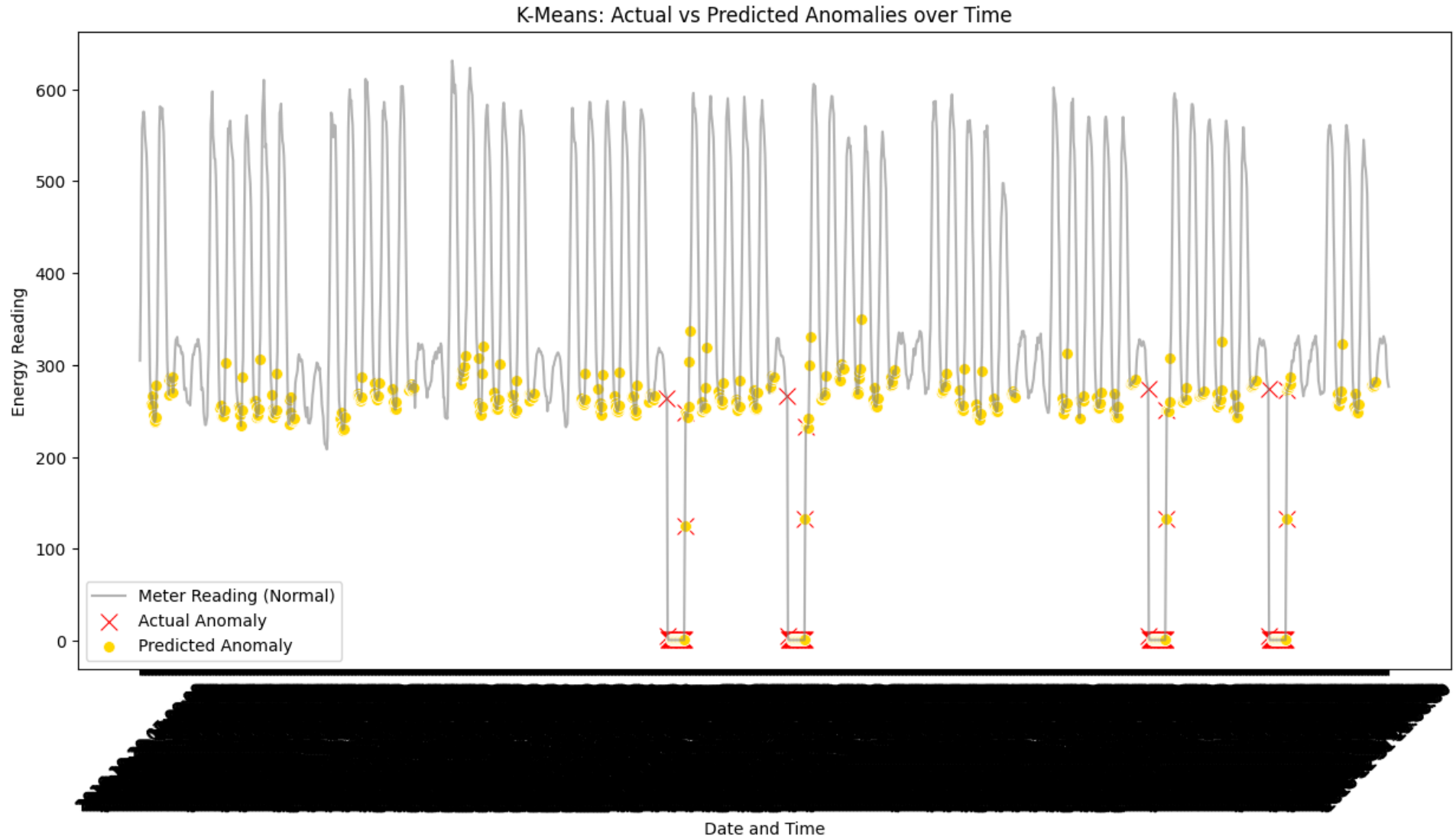
Experimental Results



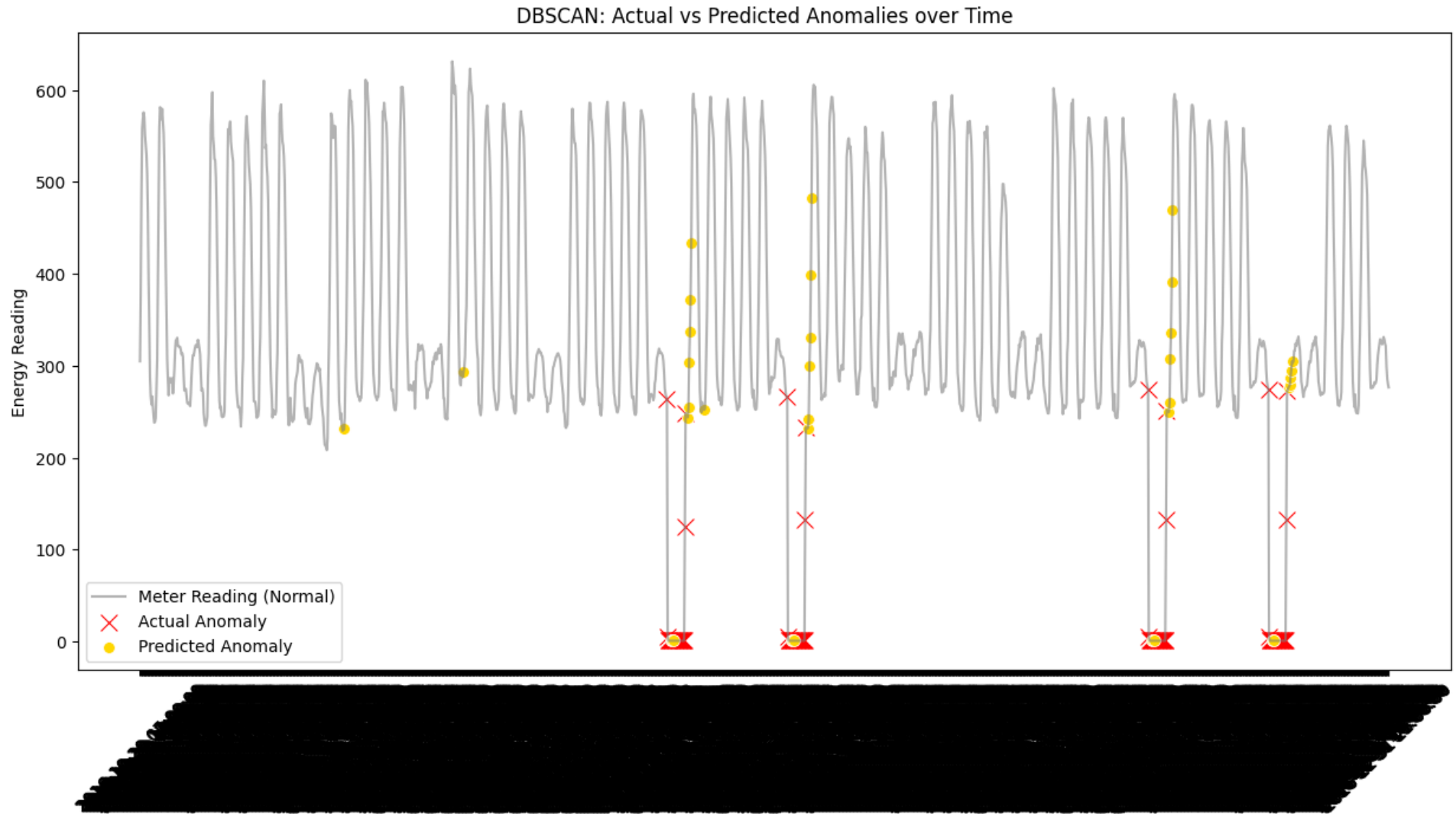
Experimental Results



Experimental Results



Experimental Results



Experimental Analysis

While I guessed that the semi-supervised method would have the best results, that was not necessarily true.

- The OC-SVM model required a fairly high mu value to be set, resulting in a lot of data to be wrongly labelled as anomalous
- The K-Means method had very varied results, that were dependent on the quality of the random LSTM compression
- The DBSCAN hybrid model, although it didn't necessarily have the best precision, recall, or F1 score did appear visually to be the best at detecting the anomaly relatively quickly after the event
- For some LSTM compressions, the DBSCAN model potentially found none of the data anomalous, but when it did, it had higher precision as compared to the other two models

Conclusion and Future Work

- **Conclusion**:
  - Inconsistent results but potentially very beneficial
  - Demonstrates that semi-supervised models are not always superior to unsupervised models
- **Main difficulty**: Lack of data
  - Poor generalization to new data because of overfitting
  - Inability to establish robust patterns
  - Hourly data is too large of an interval so that it takes a considerable amount of time for the model to detect an anomaly
- **Future Work**
  - Compare models trained and tested on other building data
  - Train and test the models on larger amount of data
  - With more detailed data, can not just detect anomalies, but possibly also determine where they might originate from
  - Use LSTM model for prediction of energy consumption for better resource management

References

https://www.kaggle.com/competitions/energy-anomaly-detection/overview

https://medium.com/@falonnekpamegan/anomaly-detection-in-sequential-data-using-lstm-autoencoder-and-kmeans-clustering-unsupervised-6c6f27149358

https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/

https://arxiv.org/abs/2305.00735

https://www.sciencedirect.com/science/article/abs/pii/S1568494621003665