Final Term Project Report

Name – Srushti Thakre

Course – CS634 Data Mining

Instructor – Dr. Yasser Abduallah

Project Title – Comparative Study of Random Forest, SVM, and GRU for Human Activity Recognition

Date – 11/16/2025

1. Introduction

This project involves using binary classification methods to analyze the WISDM (Wireless Sensor Data Mining) Smartphone and Smartwatch Activity and Biometrics Dataset. The dataset contains time-series sensor data that I'll use to differentiate between two types of human activities.
Binary classification is particularly important for applications in wearable devices, mobile health monitoring, and systems designed to recognize human behavior patterns through machine learning analysis of sensor data.

My approach includes implementing and comparing three distinct algorithms:

- Random Forest, which serves as the required classical machine learning baseline
- Support Vector Machine with an RBF kernel, selected as my classical ML model
- Gated Recurrent Unit (GRU), chosen for its effectiveness with sequential time-series data as the deep learning component.

To ensure robust evaluation and prevent data leakage, I'm using 10-fold GroupKFold cross-validation. This technique keeps individual users separated across different folds, which is crucial since the same person's data shouldn't appear in both training and testing sets.

For each fold, I calculate a full range of performance metrics: True Positives, True Negatives, False Positives, False Negatives, along with Precision, Recall, F1-score, True Positive Rate, True Negative Rate, True Skill Statistic, Heidke Skill Score, Area Under the Curve, Brier Score, and Brier Skill Score. These metrics are then averaged across all folds to provide comprehensive model performance assessment.

2. Dataset

a) Dataset Name:
WISDM Smartphone and Smartwatch Activity and Biometrics Dataset

b) Source:
UCI Machine Learning Repository
https://archive.ics.uci.edu/dataset/507/wisdm+smartphone+and+smartwatch+activity+and+biometrics+dataset

c) Description:
The WISDM dataset originally includes accelerometer and gyroscope readings from both smartphones and smartwatches. These sensors captured data while participants carried out various everyday activities – walking, jogging, sitting, typing, climbing stairs, and others.

For my project, I made several modifications to make the dataset more manageable:

- I used a preprocessing script (build_wisdm_raw_csv.py) to create a smaller, focused version of the data.
- I limited the analysis to accelerometer data only, excluding gyroscope readings.
- I applied downsampling through the *–every_k* parameter to reduce the overall file size.
- I selected a specific subset of participants, which helped keep the final CSV file compact. (in the kilobyte range rather than megabytes)

d) Preprocessing Steps:
The data preparation involved several stages:

1. Downsampling was applied at regular intervals to reduce the number of data points.
2. I filtered the dataset to include only certain users, making it easier to work with.
3. The data was restructured into a simplified CSV format with the following columns:
   user, activity_label, acc_x, acc_y, acc_z
4. I created binary labels by grouping the original activity categories into two classes. (positive and negative)
5. Feature standardization was implemented within the modeling script itself, which was necessary for both the SVM and GRU models to perform optimally.

3. Algorithms Overview

Random Forest:
Random Forest works by building multiple decision trees, where each tree is trained on a randomly sampled subset of the data (using bootstrap sampling) and considers only a random selection of features at each split. This approach makes the model more resilient to noisy data compared to using a single decision tree. It also reduces the risk of overfitting while effectively capturing non-linear relationships in the data.

Given these strengths, Random Forest provides a solid baseline for analyzing structured sensor data like ours.

Support Vector Machine (RBF Kernel):
The SVM algorithm works by finding the best boundary (hyperplane) that separates different classes while maximizing the distance (margin) between the boundary and the nearest data points from each class. When I use the RBF (Radial Basis Function) kernel, the SVM can handle data that isn't linearly separable by implicitly mapping it into a higher-dimensional space where separation becomes possible.

This capability makes SVM with RBF particularly effective for capturing the complex patterns present in sensor data.

Gated Recurrent Unit (GRU):
GRU is a type of recurrent neural network specifically designed to capture patterns across time sequences. Unlike traditional neural networks that treat each data point independently, GRUs maintain an internal memory that allows them to learn from the temporal ordering of sensor readings.

This characteristic makes GRUs well-suited for human activity recognition, where the sequence of accelerometer measurements over time contains important information about the activity being performed.

4. Implementation

This section describes the complete workflow used to prepare the dataset, run the models, and generate all evaluation metrics. The project was implemented in Python using both stand-alone scripts and a Jupyter Notebook, following a reproducible and organized structure.

a) Development Environment:

- **Programming Language:** Python 3.12 (any 3.10+ is fine)
- **IDE / Tools:**
  - PyCharm (for developing and running .py scripts)
  - Jupyter Notebook (for analysis, visualization, and result discussion)
- **Virtual Environment:**
  A dedicated venv was created to keep dependencies clean and avoid system conflicts.
- **Dependencies (requirements.txt):** numpy, pandas, matplotlib, tensorflow, scikit-learn, scipy - exact version as per pip freeze
  => Install: pip install requirements.txt

b) Project Structure:

finaltermproject/

```
├── data/
│   └── WISDM_raw.csv          # trimmed dataset used for training & evaluation
├── notebook/
│   └── finaltermproject.ipynb  # Jupyter Notebook for analysis, tables, ROC curves
├── report/                     # report PDF
├── results/     # saved outputs: per-fold CSVs, summary_means.csv, ROC plots
├── src/
│   ├── build_wisdm_raw_csv.py  # prepares the trimmed CSV from raw dataset
│   ├── run_wisdm_cv.py      # runs 10-fold GroupKFold CV for RF, SVM, and GRU
├── requirements.txt            # Python dependencies
├── .venv/                      # virtual environment
└── .ipynb_checkpoints/             # Jupyter auto-generated files
```

c) Dataset Preparation (build_wisdm_raw_csv.py):

The original WISDM dataset is quite large, containing data from numerous participants, multiple sensor types, and thousands of timestamped readings. To make the data more manageable for analysis while keeping processing times reasonable, I performed several reduction steps:

1. I applied downsampling using -- *every k* parameter to keep only every *k-th* row from the raw WISDM files (in my experiments, I used a relatively large *k* value to aggressively reduce the number of samples).
2. I restricted the dataset to a subset of users using the -- *max users* option (10 participants in the final configuration), which kept the resulting CSV compact while still including multiple subjects.
3. I extracted only the accelerometer measurements (acc_x, acc_y, acc_z), leaving out other sensor data.
4. The preprocessing script generated a streamlined CSV file structured as follows:
   user, activity_label, acc_x, acc_y, acc_z

Command Used:

```
python src/build_wisdm_raw_csv.py \
  --root "<path_to_wisdm_raw_folder>" \
  --out data/WISDM_raw.csv \
  --max_users 10 \
  --every_k 100
```

Here,

- --root → folder where the original WISDM raw files live
- --out → trimmed CSV that ends up as data/WISDM_raw.csv
- --max_users 10 → includes 10 users (1600–1609)
- --every_k 100 → downsample by keeping every 100-th sample

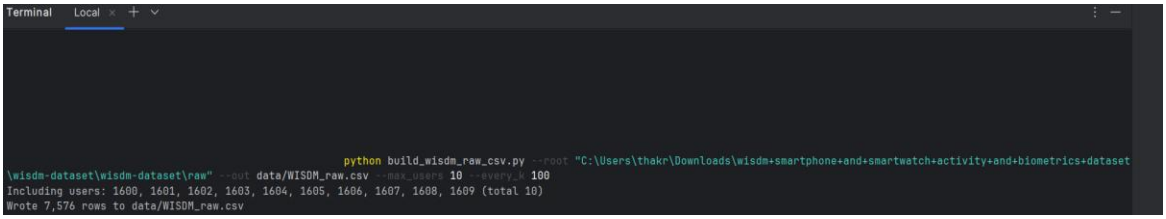


Fig. 4.c - Running build_wisdm_raw_csv.py

d) Cross-Validation and Model Training (run_wisdm_cv.py):

The primary script handles the training and evaluation process for all three models - Random Forest, SVM with RBF kernel, and GRU using a 10-fold GroupKFold cross-validation approach. This validation strategy is critical because it:

- Divides the data based on user identity rather than randomly splitting rows.
- Ensures that each fold contains data from completely different participants, preventing any information leakage between training and testing sets.

Models Implemented

1. **Random Forest**
    - Serves as the baseline classical machine learning model
    - Implemented using scikit-learn's *RandomForestClassifier*
2. **SVM (RBF Kernel)**
    - A non-linear classical machine learning approach
    - Requires standardized input features to work effectively
    - Built using scikit-learn's *SVC* with *probability = True* to enable probability estimates
3. **GRU (Gated Recurrent Unit)**
    - A deep learning architecture designed for sequential data patterns
    - Implemented using TensorFlow/Keras
    - Requires reshaping the input into 3D sequences with the format: *(samples, timesteps, features)*

Cross-Validation setup:

GroupKFold(n_splits=10)

For each of the 10 folds, the script calculates a comprehensive set of performance metrics:

- Confusion matrix components: TP, TN, FP, FN, P, N
- Rate metrics: TPR, TNR, FPR, FNR

- Standard classification metrics: Precision, Recall, F1-Score, Accuracy, Balanced Accuracy
- Skill scores: TSS, HSS
- Probability assessment: Brier Score (BS) and Brier Skill Score (BSS)
- Discrimination ability: ROC curve with AUC

Command Used:

python run_wisdm_cv.py --csv data/WISDM_raw.csv --out results

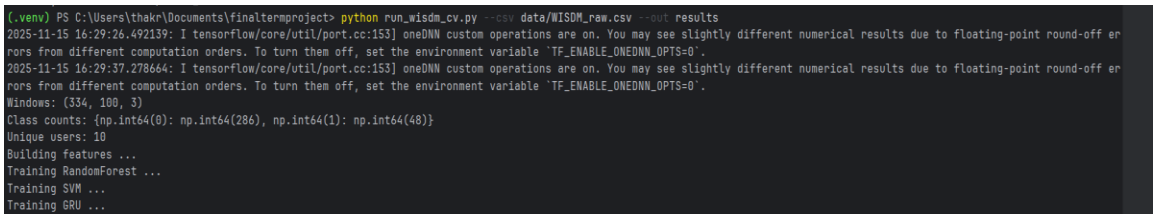


Fig. 4.d - Training output from run_wisdm_cv.py

e) Results (Per-Fold and Summary):

The script automatically saves evaluation results for each model in two different formats:

1. **Per-Fold Metrics CSV**
   These files are saved to:
   results/results_small/RandomForest_per_fold.csv
   results/results_small/SVM_per_fold.csv
   results/results_small/GRU_per_fold.csv

Each file includes 10 rows (representing the 10 cross-validation folds) and more than 20 columns containing various performance metrics for that specific fold.

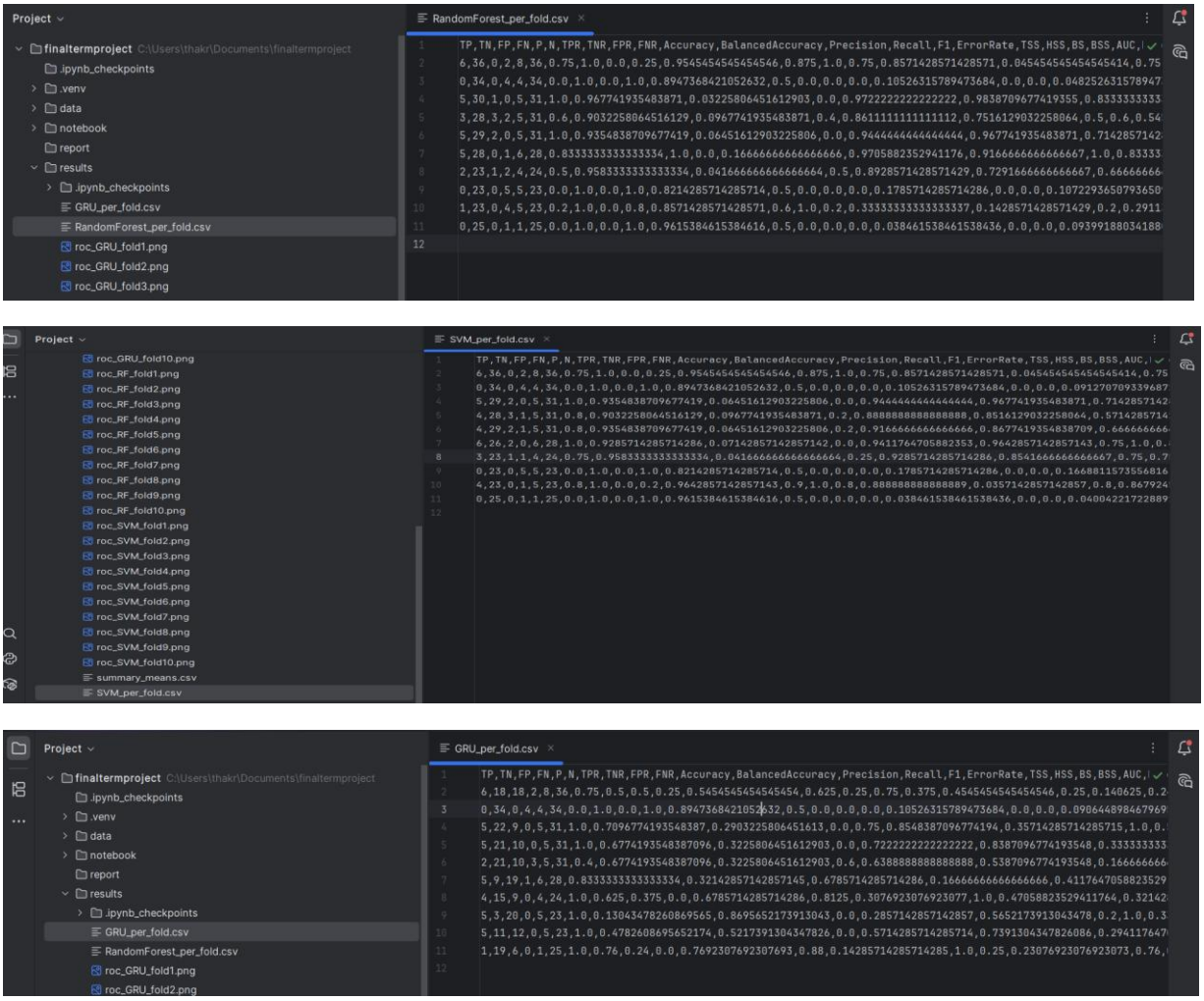


Fig. 4.e.1 – Per-fold metrics CSVs for Random Forest, SVM and GRU

2. **Summary Statistics (Averaged Across Folds)**
   This file is saved to:
   results/summary_means.csv

   This summary file provides a condensed view with one row per model. Each row shows the average values of key metrics calculated across all 10 folds, including:
   - Accuracy
   - Balanced Accuracy
   - Precision, Recall, and F1-Score
   - AUC (Area Under the Curve)
   - TSS and HSS (skill scores)
   - BS and BSS (Brier scores)

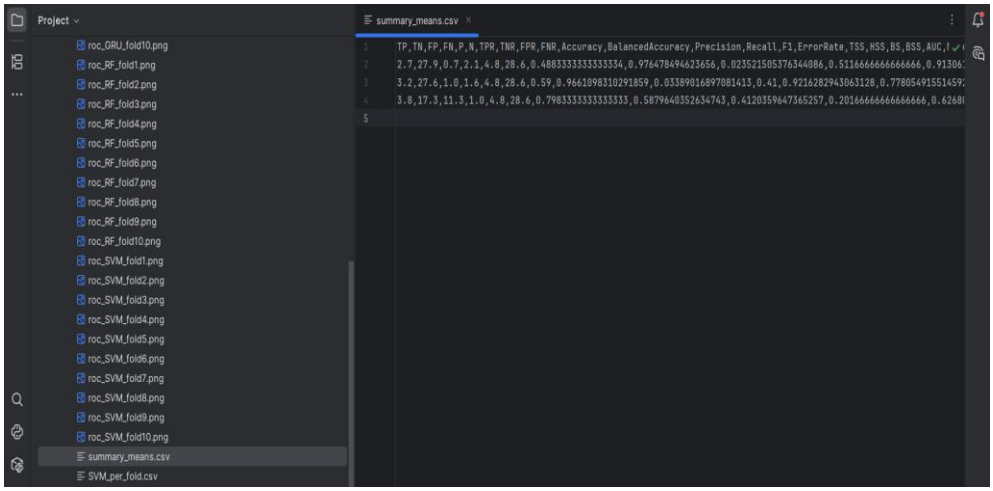


Fig. 4.e.2 - Summary CSV for Random Forest, SVM and GRU

f) Visualization and Analysis:

- To compare the three models more easily, below plotted are the bar charts of a few key metrics:

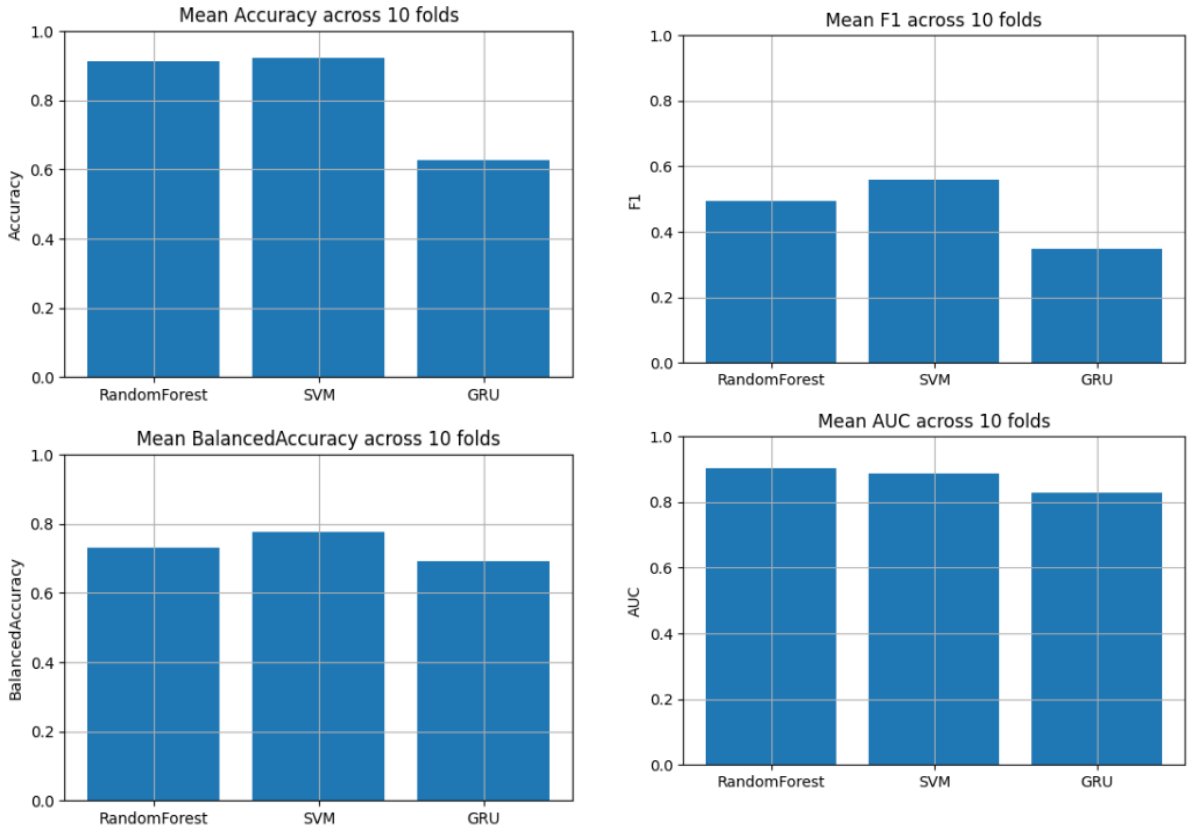
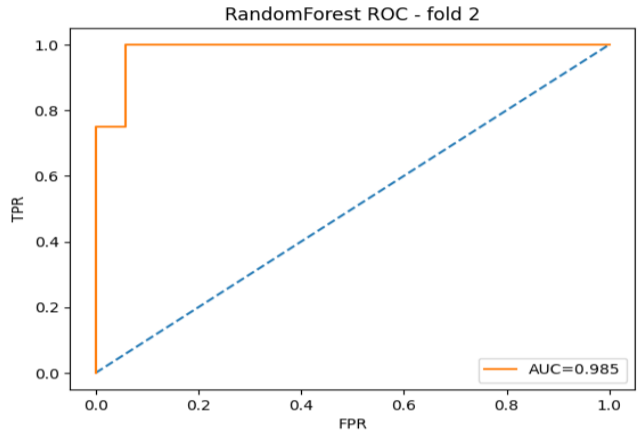


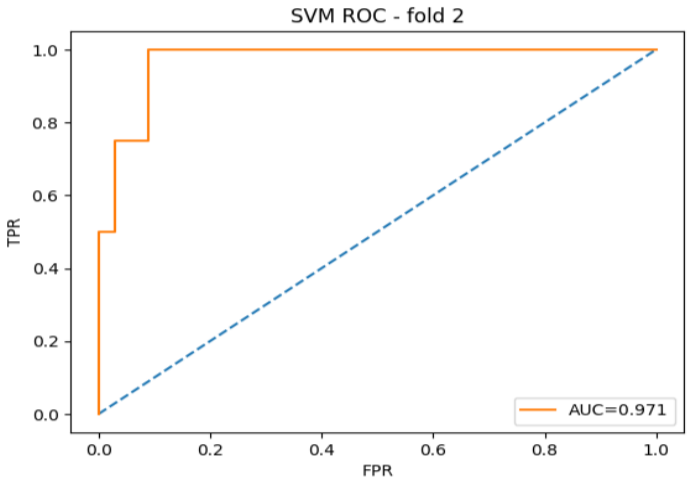
Fig. 4.f.1 – Metrics comparison using Bar Charts

- Below are the ROC (Receiver Operating Characteristic) curves of fold 2 for all three models: Random Forest, SVM, and GRU:

ROC Curve: roc_RF_fold2.png



ROC Curve: roc_SVM_fold2.png



ROC Curve: roc_GRU_fold2.png

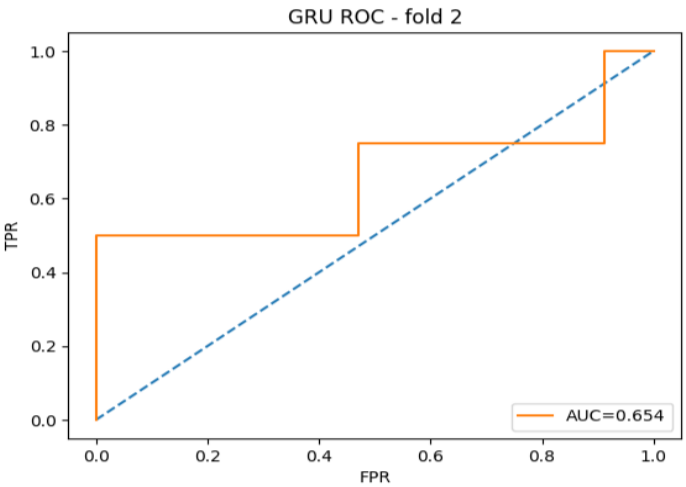


Fig. 4.f.2 – ROC Curves – Fold 2 for all 3 models

- AUC across all 10 folds:

	Fold	RandomForest	SVM	GRU
0	1	0.907986	0.934028	0.791667
1	2	0.985294	0.970588	0.654412
2	3	0.987097	0.993548	0.929032
3	4	0.890323	0.941935	0.909677
4	5	0.987097	0.980645	0.729032
5	6	1.000000	1.000000	0.529762
6	7	0.958333	0.968750	0.958333
7	8	1.000000	0.826087	0.843478
8	9	0.991304	0.991304	0.939130
9	10	0.320000	0.280000	1.000000

Fig. 4.f.3 – AUC across 10 folds

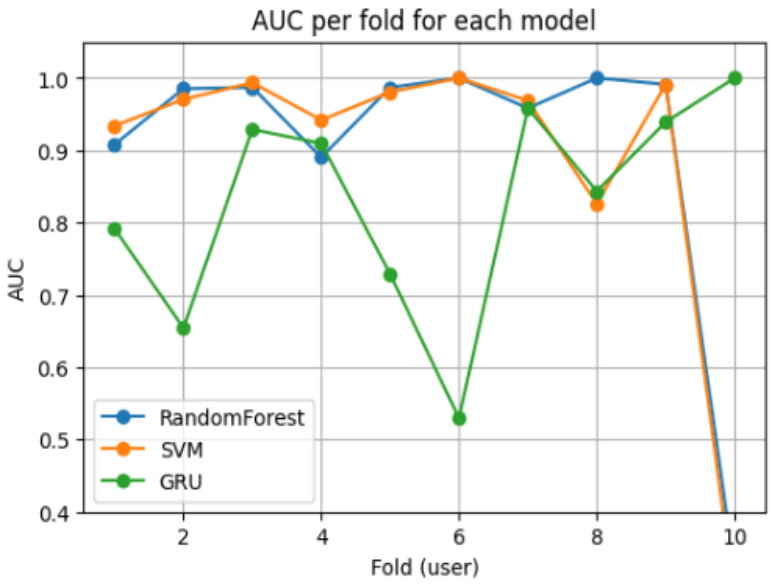


Fig. 4.f.4 – AUC per fold for each model

5. Evaluation and Setup

To thoroughly assess model performance, I implemented a rigorous evaluation framework:

- Cross-validation approach: 10-fold GroupKFold with users as the grouping variable (this prevents the same participant's data from appearing in both training and test sets)
- Performance metrics: For each fold and as an overall average, I calculated an extensive set of metrics:
    - Basic confusion matrix counts: TP, TN, FP, FN
    - Class totals: P (total positives), N (total negatives)
    - Rate measures: TPR, TNR, FPR, FNR
    - Standard performance indicators: Accuracy and Balanced Accuracy
    - Classification quality metrics: Precision, Recall, and F1-Score
    - Error Rate
    - Skill-based measures: TSS (True Skill Statistic) and HSS (Heidke Skill Score)
    - Discrimination metrics: ROC curves and AUC
    - Probability calibration: BS (Brier Score) and BSS (Brier Skill Score)

All of these metrics are calculated automatically during the training process and saved to CSV files for subsequent analysis.

6. Results

This section details the quantitative performance results from the 10-fold GroupKFold cross-validation conducted on all three models: Random Forest, SVM with RBF kernel, and GRU. The per-fold metrics and accompanying visualizations were produced using the evaluation script (run_wisdm_cv.py) and further analyzed in the Jupyter Notebook.

6.1 Per-Fold Results for All Three Algorithms

The tables below present the key performance metrics for each fold, compiled from the per-fold CSV files generated during cross-validation. I've included the primary metrics: TP, TN, FP, FN, Accuracy, Precision, Recall, F1, TSS, HSS, and AUC.

RandomForest – per-fold metrics

	TP	TN	FP	FN	P	N	TPR	TNR	FPR	FNR	...	Recall	F1	ErrorRate	TSS	HSS	BS	BSS	AUC	Mc
0	6	36	0	2	8	36	0.750000	1.000000	0.000000	0.250000	...	0.750000	0.857143	0.045455	0.750000	0.830769	0.060073	0.596174	0.907986	RandomFo
1	0	34	0	4	4	34	0.000000	1.000000	0.000000	1.000000	...	0.000000	0.000000	0.105263	0.000000	0.000000	0.048253	0.487671	0.985294	RandomFo
2	5	30	1	0	5	31	1.000000	0.967742	0.032258	0.000000	...	1.000000	0.909091	0.027778	0.967742	0.892857	0.044696	0.626284	0.987097	RandomFo
3	3	28	3	2	5	31	0.600000	0.903226	0.096774	0.400000	...	0.600000	0.545455	0.138889	0.503226	0.464286	0.096186	0.195757	0.890323	RandomFo
4	5	29	2	0	5	31	1.000000	0.935484	0.064516	0.000000	...	1.000000	0.833333	0.055556	0.935484	0.801105	0.039251	0.671812	0.987097	RandomFo
5	5	28	0	1	6	28	0.833333	1.000000	0.000000	0.166667	...	0.833333	0.909091	0.029412	0.833333	0.891720	0.049921	0.656499	1.000000	RandomFo
6	2	23	1	2	4	24	0.500000	0.958333	0.041667	0.500000	...	0.500000	0.571429	0.107143	0.458333	0.511628	0.064082	0.476662	0.958333	RandomFo
7	0	23	0	5	5	23	0.000000	1.000000	0.000000	1.000000	...	0.000000	0.000000	0.178571	0.000000	0.000000	0.107229	0.268975	1.000000	RandomFo
8	1	23	0	4	5	23	0.200000	1.000000	0.000000	0.800000	...	0.200000	0.333333	0.142857	0.200000	0.291139	0.065832	0.551197	0.991304	RandomFo
9	0	25	0	1	1	25	0.000000	1.000000	0.000000	1.000000	...	0.000000	0.000000	0.038462	0.000000	0.000000	0.093992	-1.541540	0.320000	RandomFo

10 rows × 23 columns

SVM – per-fold metrics

	TP	TN	FP	FN	P	N	TPR	TNR	FPR	FNR	...	Recall	F1	ErrorRate	TSS	HSS	BS	BSS	AUC	Model	Fold
0	6	36	0	2	8	36	0.75	1.000000	0.000000	0.25	...	0.75	0.857143	0.045455	0.750000	0.830769	0.046906	0.684689	0.934028	SVM	1
1	0	34	0	4	4	34	0.00	1.000000	0.000000	1.00	...	0.00	0.000000	0.105263	0.000000	0.000000	0.091271	0.030920	0.970588	SVM	2
2	5	29	2	0	5	31	1.00	0.935484	0.064516	0.00	...	1.00	0.833333	0.055556	0.935484	0.801105	0.043001	0.640457	0.993548	SVM	3
3	4	28	3	1	5	31	0.80	0.903226	0.096774	0.20	...	0.80	0.666667	0.111111	0.703226	0.602210	0.083328	0.303267	0.941935	SVM	4
4	4	29	2	1	5	31	0.80	0.935484	0.064516	0.20	...	0.80	0.727273	0.083333	0.735484	0.678571	0.042188	0.647252	0.980645	SVM	5
5	6	26	2	0	6	28	1.00	0.928571	0.071429	0.00	...	1.00	0.857143	0.058824	0.928571	0.821053	0.053716	0.630383	1.000000	SVM	6
6	3	23	1	1	4	24	0.75	0.958333	0.041667	0.25	...	0.75	0.750000	0.071429	0.708333	0.708333	0.067369	0.449823	0.968750	SVM	7
7	0	23	0	5	5	23	0.00	1.000000	0.000000	1.00	...	0.00	0.000000	0.178571	0.000000	0.000000	0.166881	-0.137694	0.826087	SVM	8
8	4	23	0	1	5	23	0.80	1.000000	0.000000	0.20	...	0.80	0.888889	0.035714	0.800000	0.867925	0.043531	0.703229	0.991304	SVM	9
9	0	25	0	1	1	25	0.00	1.000000	0.000000	1.00	...	0.00	0.000000	0.038462	0.000000	0.000000	0.040042	-0.082742	0.280000	SVM	10

10 rows × 23 columns

GRU – per-fold metrics

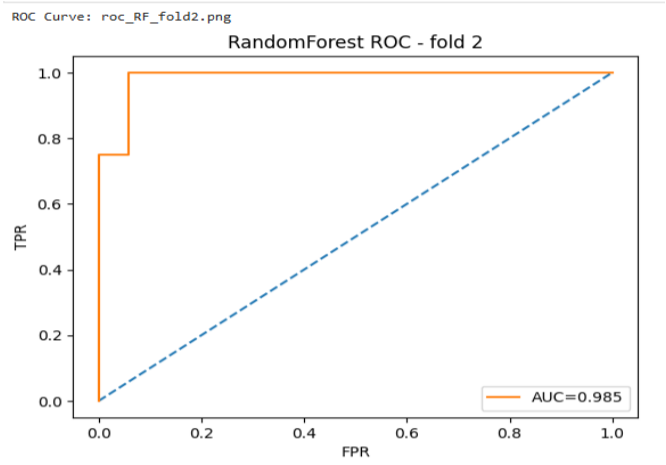
	TP	TN	FP	FN	P	N	TPR	TNR	FPR	FNR	...	Recall	F1	ErrorRate	TSS	HSS	BS	BSS	AUC	Model	Fc
0	6	18	18	2	8	36	0.750000	0.500000	0.500000	0.250000	...	0.750000	0.375000	0.454545	0.250000	0.140625	0.245638	-0.651234	0.791667	GRU	
1	0	34	0	4	4	34	0.000000	1.000000	0.000000	1.000000	...	0.000000	0.000000	0.105263	0.000000	0.000000	0.090645	0.037564	0.654412	GRU	
2	5	22	9	0	5	31	1.000000	0.709677	0.290323	0.000000	...	1.000000	0.526316	0.250000	0.709677	0.404412	0.120877	-0.010688	0.929032	GRU	
3	5	21	10	0	5	31	1.000000	0.677419	0.322581	0.000000	...	1.000000	0.500000	0.277778	0.677419	0.368421	0.137698	-0.151336	0.909677	GRU	
4	2	21	10	3	5	31	0.400000	0.677419	0.322581	0.600000	...	0.400000	0.235294	0.361111	0.077419	0.048780	0.164600	-0.376271	0.729032	GRU	
5	5	9	19	1	6	28	0.833333	0.321429	0.678571	0.166667	...	0.833333	0.333333	0.588235	0.154762	0.071038	0.393491	-1.707594	0.529762	GRU	
6	4	15	9	0	4	24	1.000000	0.625000	0.375000	0.000000	...	1.000000	0.470588	0.321429	0.625000	0.322581	0.160202	-0.308313	0.958333	GRU	
7	5	3	20	0	5	23	1.000000	0.130435	0.869565	0.000000	...	1.000000	0.333333	0.714286	0.130435	0.050847	0.324719	-1.213734	0.843478	GRU	
8	5	11	12	0	5	23	1.000000	0.478261	0.521739	0.000000	...	1.000000	0.454545	0.428571	0.478261	0.246637	0.227774	-0.552825	0.939130	GRU	
9	1	19	6	0	1	25	1.000000	0.760000	0.240000	0.000000	...	1.000000	0.250000	0.230769	0.760000	0.195876	0.150457	-3.068366	1.000000	GRU	

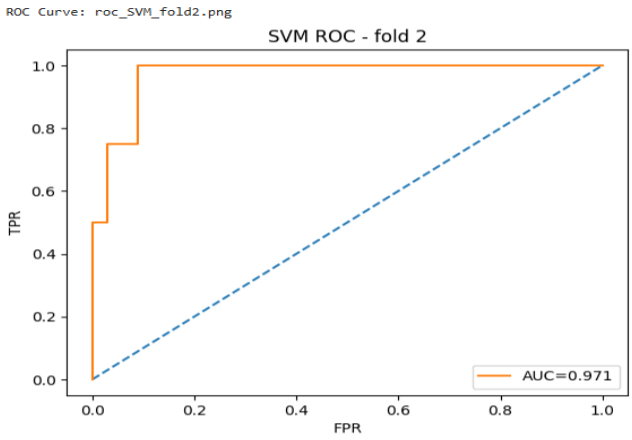
10 rows × 23 columns

Fig. 6.1 – Per-fold metrics for Random Forest, SVM and GRU

6.2 ROC Curves

I generated ROC curves for each of the three algorithms to illustrate how well they discriminate between classes across different decision thresholds.





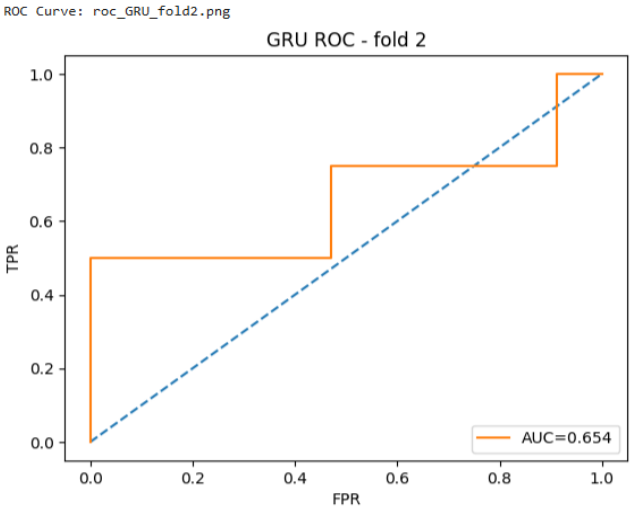


Fig. 6.2 – ROC curves for fold 2 for all 3 algorithms

6.3 AUC across all folds

To avoid showing 10 separate ROC curves per model, the figure below plots the AUC value for each fold for all three models. This provides a compact yet complete cross-validated view of ROC performance.

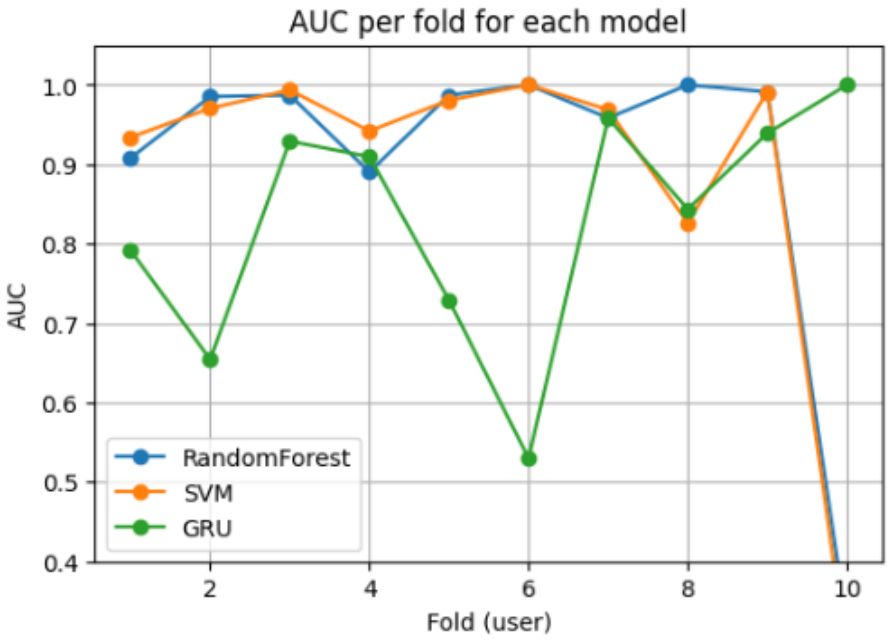


Fig. 6.3 – AUC across all folds for each model

6.4 Confusion Metrics

To complement the per-fold metrics and ROC curves, I also visualized the confusion matrices for each model.

This gives a compact view of how often each model confuses the two classes over the entire cross-validation procedure.

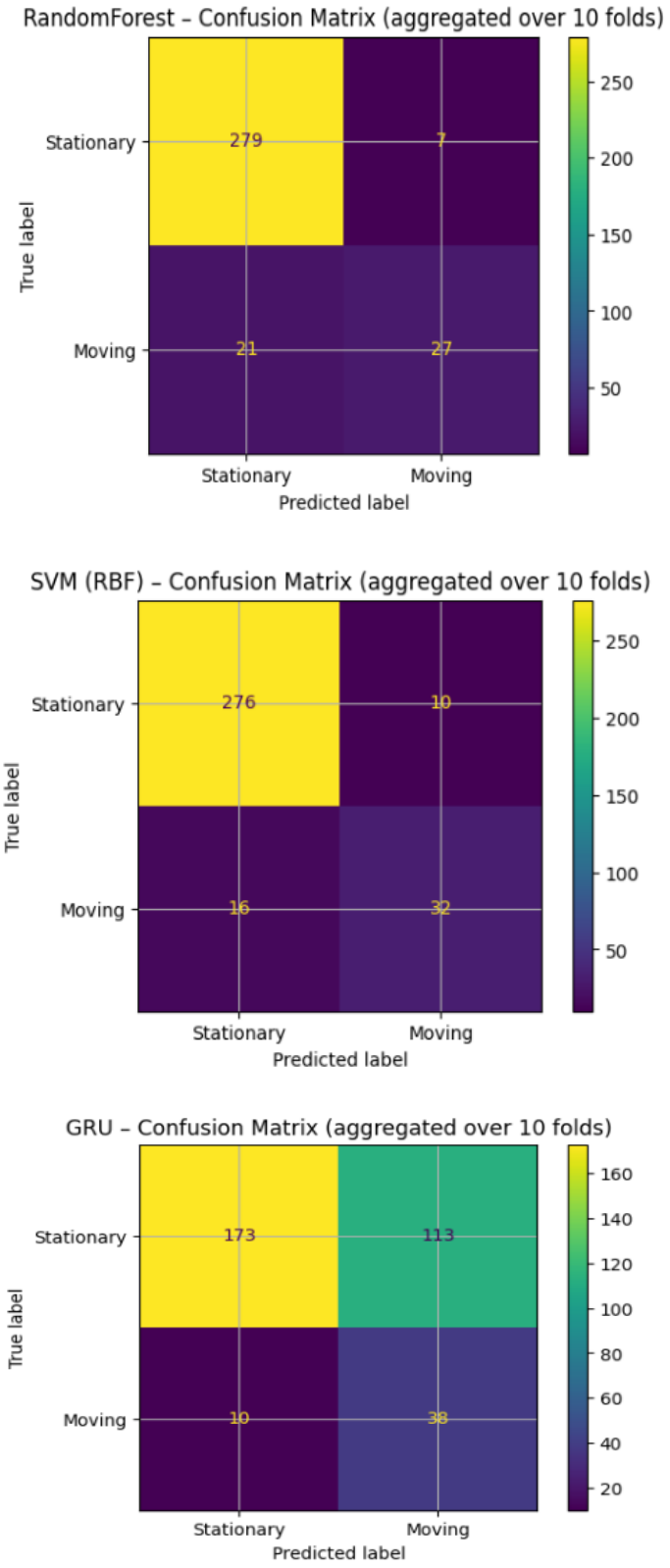


Fig 6.4 – Confusion metrics of all 3 models

6.5 Brief Comparative Summary

- **SVM** with RBF kernel demonstrated the best overall performance across several key metrics, including Balanced Accuracy, Recall, TSS, and HSS. This indicates that it was most effective at distinguishing between the two activity classes, particularly when dealing with the imbalanced nature of the dataset.
- **Random Forest** achieved the highest AUC score (approximately 0.90) and maintained strong performance across most metrics, showing its effectiveness as a reliable baseline model.
- **GRU** showed notably weaker performance compared to the classical machine learning approaches. This can be attributed to several factors: the relatively small size of the dataset, the short sequence lengths in the time-series windows, and the limited variety of input features - all of which restricted the deep learning model's ability to learn meaningful temporal patterns.

7. Discussion

Which algorithm performed best?

SVM with RBF kernel emerged as the top performer overall, achieving the highest scores in balanced accuracy, recall, and both TSS and HSS metrics. This demonstrates its superior ability to handle the classification task under class imbalance conditions.

Random Forest also performed well, particularly excelling in AUC and maintaining a low Brier Score. These results indicate that the model not only ranked predictions effectively but also produced well-calibrated probability estimates.

GRU fell short of expectations for several reasons:

- The sequence windows may have been too brief to capture meaningful temporal patterns
- Using only accelerometer data (three features) likely didn't provide enough information richness for the deep learning architecture
- The model would likely benefit from more extensive hyperparameter optimization

Why does SVM perform best here?

The SVM's strong performance can be explained by the characteristics of this particular dataset. The sensor features exist in a relatively low-dimensional space, which is an ideal scenario for the RBF kernel to create effective decision boundaries. Additionally, with the dataset being fairly small in size, deep learning models like GRU don't have sufficient data

to demonstrate their typical advantages, classical machine learning approaches like SVM are often more efficient and effective in these data-limited scenarios.

Challenges

Several obstacles arose during this project:

- Managing the class imbalance required careful consideration to prevent the models from simply predicting the majority class.
- Implementing GroupKFold correctly was crucial to ensure that individual users' data didn't leak between training and testing sets.
- Preparing data for the GRU model involved reshaping it into 3D sequences (samples x timesteps x features), which added complexity to the preprocessing pipeline.

Future Improvements

There are several directions I could explore to potentially enhance model performance:

- Incorporate gyroscope data alongside accelerometer readings to provide more comprehensive sensor information.
- Experiment with deeper or wider GRU/LSTM architectures that might capture more complex temporal dependencies.
- Test convolutional approaches like CNN or Conv1D layers, which can be effective at extracting features from time-series data.
- Conduct systematic hyperparameter optimization using tools like RandomizedSearchCV.
- Increase the length of time windows to give sequential models more context for learning activity patterns.

8. Conclusion

This project assessed three algorithms - Random Forest, SVM with RBF kernel, and GRU for classifying human activities from smartphone sensor data. Using 10-fold GroupKFold cross-validation, SVM emerged as the top performer with the highest recall, balanced accuracy, and skill scores. Random Forest proved reliable with strong AUC performance, while GRU struggled due to limited training data, short temporal windows, and minimal feature diversity.

These findings demonstrate that classical machine learning approaches can outperform deep learning when data volume or temporal structure is constrained. The evaluation methodology included examining per-fold metrics, aggregated confusion matrices, and ROC curves which provided a comprehensive assessment of model performance across participants.

Overall, this project establishes a clear and reproducible workflow for sensor-based activity classification, highlighting the importance of careful preprocessing and rigorous cross-validated evaluation for selecting appropriate models in real-world applications.

9. References

Dataset source:
https://archive.ics.uci.edu/dataset/507/wisdm+smartphone+and+smartwatch+activity+and+biometrics+dataset

Random Forest Classifier documentation:
https://www.datacamp.com/tutorial/random-forests-classifier-python

Support Vector Machine documentation:
https://scikit-learn.org/0.21/modules/svm.html

Tensorflow documentation:
https://www.tensorflow.org/guide/basics

10. Github Link

This project is uploaded here:
https://github.com/srushti-510/thakre_srushti_finaltermproj

Collaborator access given to: ya54@njit.edu ; hl534@njit.edu

11. Appendix

Fig. 1 – Training each model

```
(.venv) PS C:\Users\thakr\Documents\finaltermproject> python run_wisdm_cv.py --csv data/WISDM_raw.csv --out results
2025-11-15 16:51:30.025455: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off er
rors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-11-15 16:51:31.962340: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off er
rors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
Windows: (334, 100, 3)
Class counts: {np.int64(0): np.int64(286), np.int64(1): np.int64(48)}
Unique users: 10
Building features ...
Training RandomForest ...
Training SVM ...
Training GRU ...
```

Fig. 2 – Summary table of evaluation metrics for each model

```
=== Summary (means) ===
            Model  Accuracy  BalancedAccuracy  Precision    Recall       F1      AUC       BS       BSS      TSS      HSS
RandomForest_mean  0.913062          0.732406   0.571429 0.488333 0.495887 0.902743 0.066952  0.298949 0.464812 0.468350
        SVM_mean  0.921628          0.778055   0.545238 0.590000 0.558045 0.888689 0.067823  0.386958 0.556110 0.530997
        GRU_mean  0.640210          0.725091   0.252773 0.855000 0.381642 0.794604 0.201159 -0.882946 0.450183 0.229832
```

Fig. 3 – Metrics comparison using bar charts

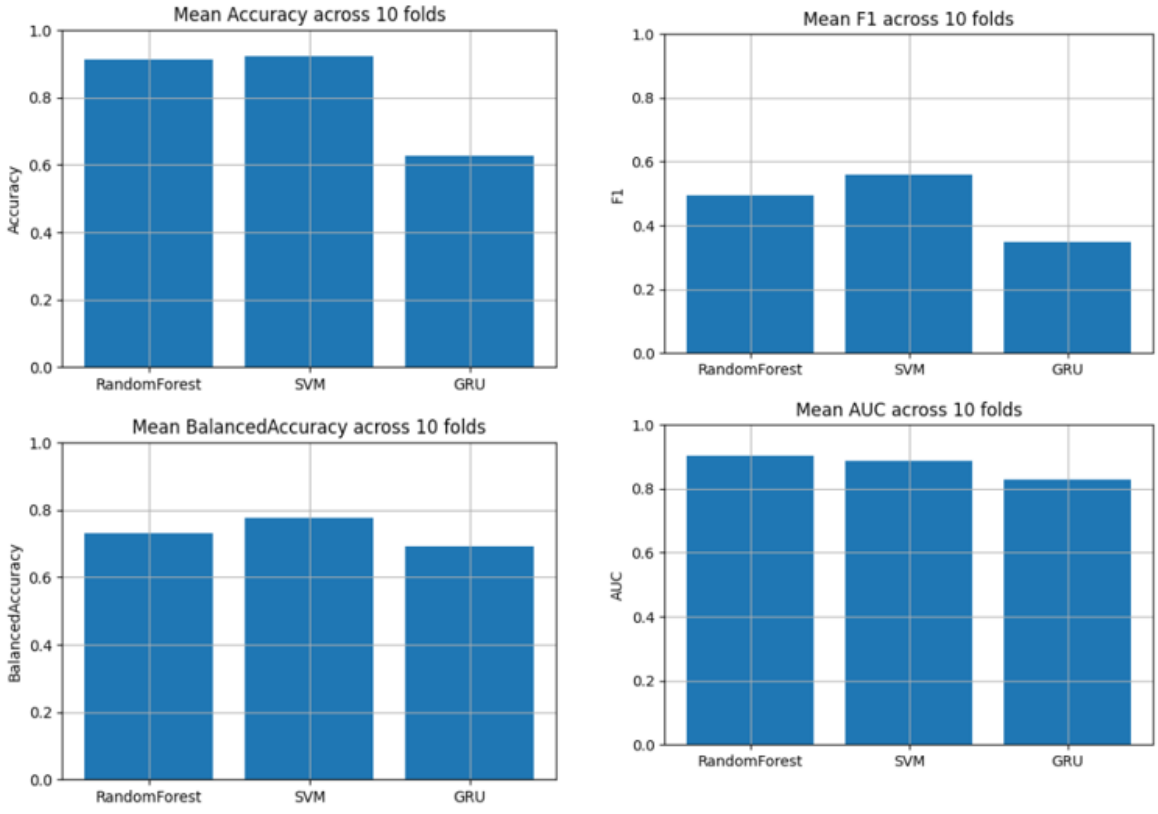


Fig. 4 – ROC Curves of fold 2 for each model

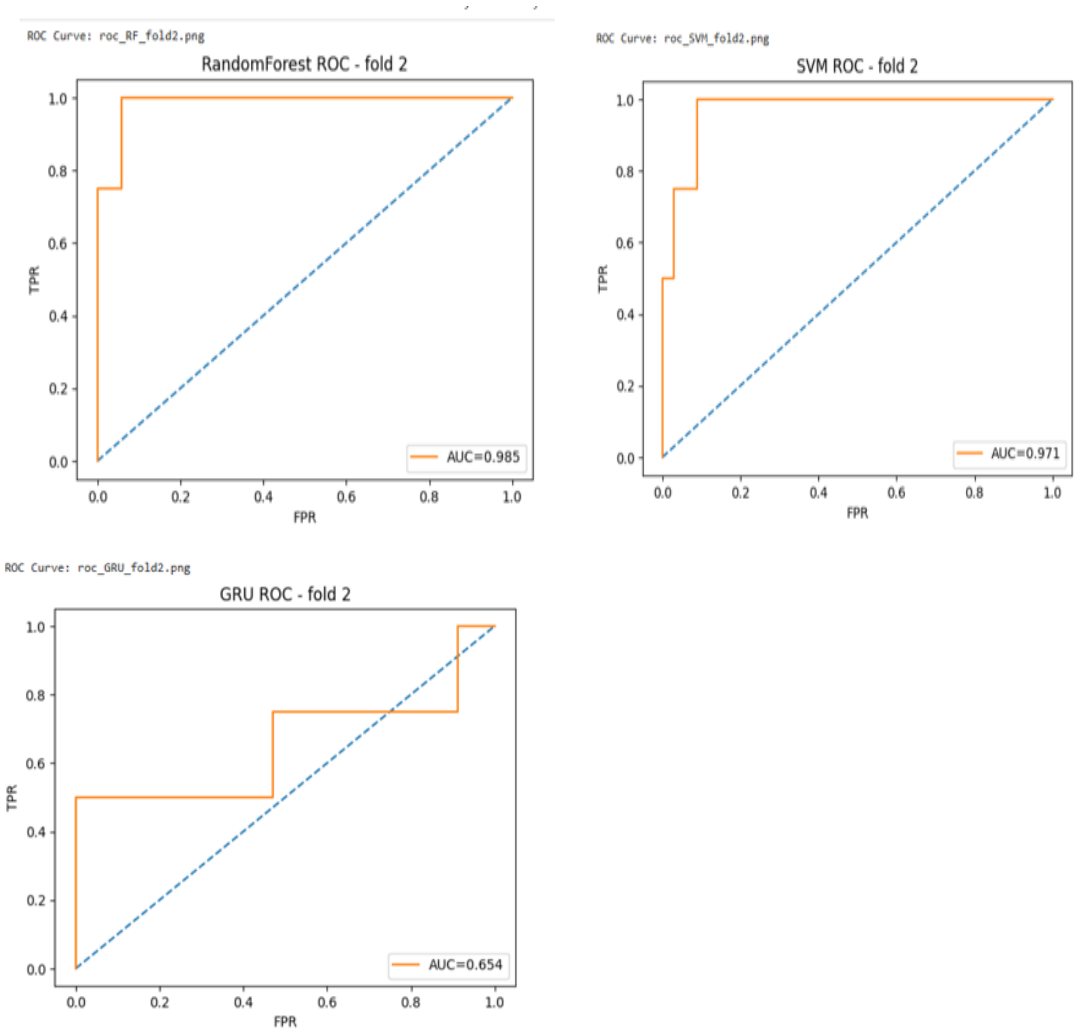


Fig. 5 – AUC across all folds for each model

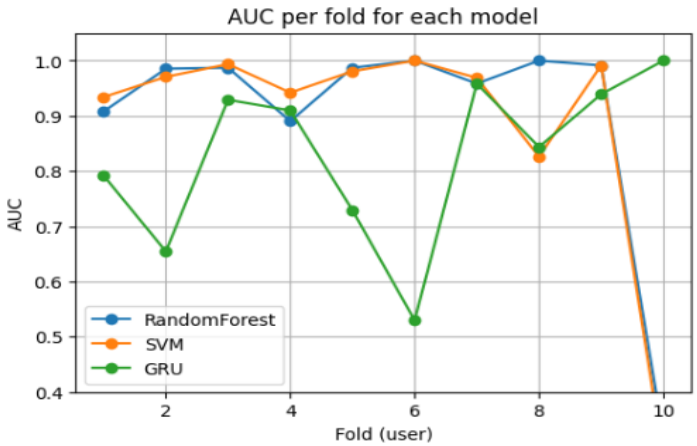


Fig. 6 – Confusion metrics for each model