Final Project Report

Course: CS 634 – Data Mining

Instructor: Dr. Yasser Abduallah

Student Name: Dathwik Kollikonda (dk649)

Project Title: Wine Quality Binary Classification Using Machine Learning and Deep Learning

Submission Date: April 6th 2025

1. Dataset Description

- Dataset: Wine Quality – Red Wine
- Source: UCI Machine Learning Repository https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv, https://archive.ics.uci.edu/dataset/186/wine+quality (There are two datasets for wine quality, I chose the red wine dataset.)
- Type: Tabular dataset with 11 physicochemical features and a quality score (0–10).
- Size: 1599 samples × 12 columns
- Target Conversion:
  - Quality ≥ 6 → Good (1)
  - Quality < 6 → Bad (0)
  - Binary column created: quality_binary

2. Algorithms Used

Algorithm	Type	Source Library
Random Forest	Ensemble ML	Scikit-learn
K-Nearest Neighbors (KNN)	Instance-based ML	Scikit-learn
GRU (Gated Recurrent Unit)	Deep Learning	TensorFlow / Keras

3. Methodology

Preprocessing:
- Converted quality scores to binary target variable.
- Features were normalized using StandardScaler.

- Data was shuffled for randomness.

Cross-Validation:
- Used StratifiedKFold with 10 splits to maintain label balance.

Model Training:
- Random Forest: n_estimators=100
- KNN: n_neighbors=5
- GRU: 32-unit GRU with dropout, sigmoid activation, trained for 20 epochs with early stopping.

Metrics Computed (Manually):
- Accuracy, Error Rate, Recall (Sensitivity), Precision, F1 Score, FPR, FNR, TSS, HSS

4. Results & Evaluation

Average Performance Over 10 Folds:

Metric	**Random Forest**	**KNN**	**GRU**
Accuracy	*0.8280*	0.7336	0.7505
Error Rate	*0.1720*	0.2664	0.2495
Recall	*0.8373*	0.7870	0.7496
Precision	*0.8415*	0.7342	0.7785
F1 Score	*0.8387*	0.7592	0.7621
TSS	*0.6546*	0.4591	0.5010
HSS	*0.6544*	0.4617	0.4997

5. Discussion

Random Forest outperformed KNN and GRU across all evaluation metrics.
This is expected as Random Forest handles noisy tabular data well, builds multiple trees, and reduces overfitting through ensembling.
KNN was fast and easy to train but showed weakness in precision due to class overlap and sensitivity to feature scale.
GRU, while competitive, is primarily designed for sequential data (e.g., time-series) and does not have a structural advantage over Random Forest in this tabular, non-temporal context.

6. How to Run the Project

Requirements:
- Python 3.8+
- Jupyter Notebook/Google Colab
- Required packages:
  - pandas
  - numpy
  - scikit-learn
  - tensorflow
  - matplotlib

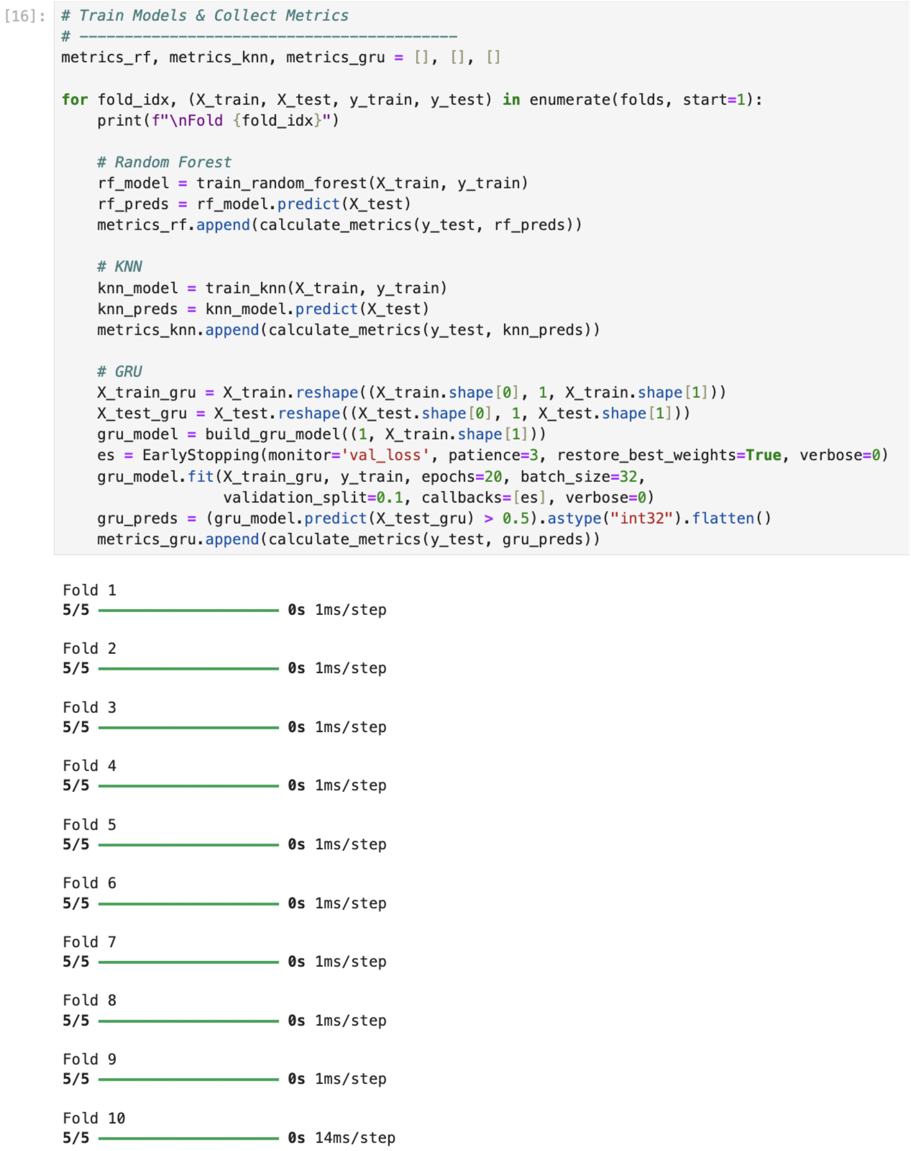
Instructions:
1. Place winequality-red.csv in the working directory.
2. Open kollikonda_dathwik_finalproject.ipynb in Jupyter.
3. Run all cells top to bottom.
4. The notebook will output fold-by-fold metrics and summary tables.

7. GitHub Repository

Repository URL: https://github.com/DathwikK/wine-red-quality

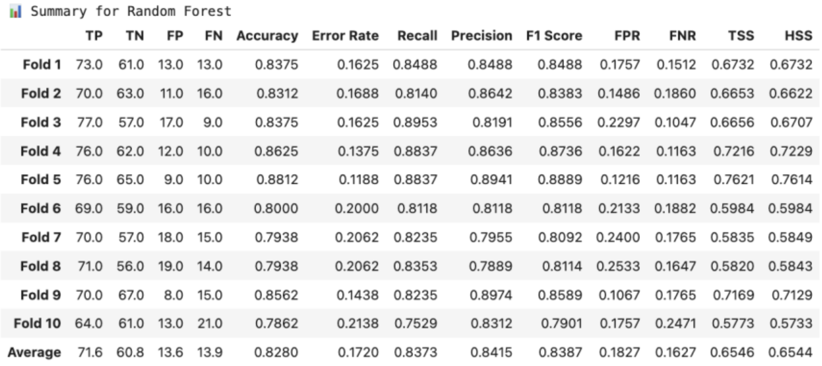
Collaborator Added: ya54@njit.edu

8. Screenshots



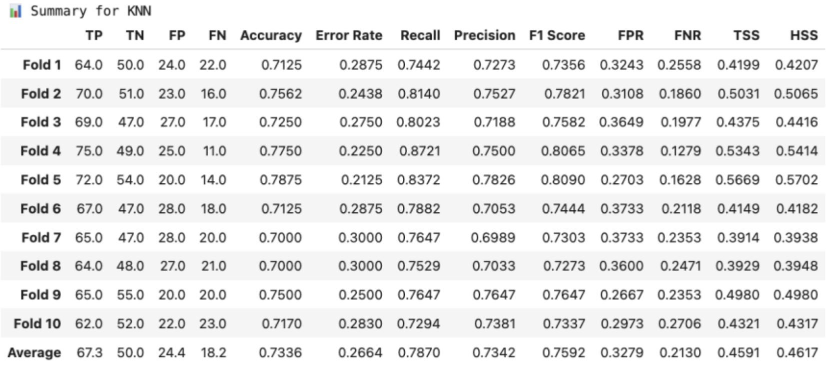
```
[16]: # Train Models & Collect Metrics
      # ------------------------------------------
      metrics_rf, metrics_knn, metrics_gru = [], [], []

      for fold_idx, (X_train, X_test, y_train, y_test) in enumerate(folds, start=1):
          print(f"\nFold {fold_idx}")

          # Random Forest
          rf_model = train_random_forest(X_train, y_train)
          rf_preds = rf_model.predict(X_test)
          metrics_rf.append(calculate_metrics(y_test, rf_preds))

          # KNN
          knn_model = train_knn(X_train, y_train)
          knn_preds = knn_model.predict(X_test)
          metrics_knn.append(calculate_metrics(y_test, knn_preds))

          # GRU
          X_train_gru = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
          X_test_gru = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
          gru_model = build_gru_model((1, X_train.shape[1]))
          es = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True, verbose=0)
          gru_model.fit(X_train_gru, y_train, epochs=20, batch_size=32,
                        validation_split=0.1, callbacks=[es], verbose=0)
          gru_preds = (gru_model.predict(X_test_gru) > 0.5).astype("int32").flatten()
          metrics_gru.append(calculate_metrics(y_test, gru_preds))
```

```
Fold 1
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step

Fold 2
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step

Fold 3
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step

Fold 4
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step

Fold 5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step

Fold 6
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step

Fold 7
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step

Fold 8
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step

Fold 9
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step

Fold 10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
```



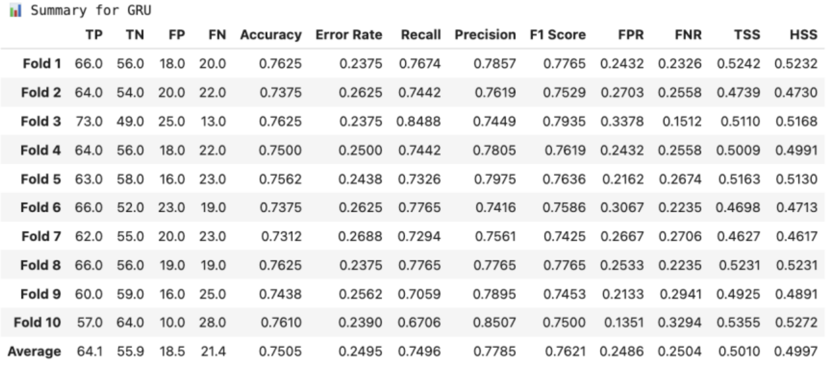
📊 Summary for Random Forest

	TP	TN	FP	FN	Accuracy	Error Rate	Recall	Precision	F1 Score	FPR	FNR	TSS	HSS
Fold 1	73.0	61.0	13.0	13.0	0.8375	0.1625	0.8488	0.8488	0.8488	0.1757	0.1512	0.6732	0.6732
Fold 2	70.0	63.0	11.0	16.0	0.8312	0.1688	0.8140	0.8642	0.8383	0.1486	0.1860	0.6653	0.6622
Fold 3	77.0	57.0	17.0	9.0	0.8375	0.1625	0.8953	0.8191	0.8556	0.2297	0.1047	0.6656	0.6707
Fold 4	76.0	62.0	12.0	10.0	0.8625	0.1375	0.8837	0.8636	0.8736	0.1622	0.1163	0.7216	0.7229
Fold 5	76.0	65.0	9.0	10.0	0.8812	0.1188	0.8837	0.8941	0.8889	0.1216	0.1163	0.7621	0.7614
Fold 6	69.0	59.0	16.0	16.0	0.8000	0.2000	0.8118	0.8118	0.8118	0.2133	0.1882	0.5984	0.5984
Fold 7	70.0	57.0	18.0	15.0	0.7938	0.2062	0.8235	0.7955	0.8092	0.2400	0.1765	0.5835	0.5849
Fold 8	71.0	56.0	19.0	14.0	0.7938	0.2062	0.8353	0.7889	0.8114	0.2533	0.1647	0.5820	0.5843
Fold 9	70.0	67.0	8.0	15.0	0.8562	0.1438	0.8235	0.8974	0.8589	0.1067	0.1765	0.7169	0.7129
Fold 10	64.0	61.0	13.0	21.0	0.7862	0.2138	0.7529	0.8312	0.7901	0.1757	0.2471	0.5773	0.5733
Average	71.6	60.8	13.6	13.9	0.8280	0.1720	0.8373	0.8415	0.8387	0.1827	0.1627	0.6546	0.6544



📊 Summary for KNN

	TP	TN	FP	FN	Accuracy	Error Rate	Recall	Precision	F1 Score	FPR	FNR	TSS	HSS
Fold 1	64.0	50.0	24.0	22.0	0.7125	0.2875	0.7442	0.7273	0.7356	0.3243	0.2558	0.4199	0.4207
Fold 2	70.0	51.0	23.0	16.0	0.7562	0.2438	0.8140	0.7527	0.7821	0.3108	0.1860	0.5031	0.5065
Fold 3	69.0	47.0	27.0	17.0	0.7250	0.2750	0.8023	0.7188	0.7582	0.3649	0.1977	0.4375	0.4416
Fold 4	75.0	49.0	25.0	11.0	0.7750	0.2250	0.8721	0.7500	0.8065	0.3378	0.1279	0.5343	0.5414
Fold 5	72.0	54.0	20.0	14.0	0.7875	0.2125	0.8372	0.7826	0.8090	0.2703	0.1628	0.5669	0.5702
Fold 6	67.0	47.0	28.0	18.0	0.7125	0.2875	0.7882	0.7053	0.7444	0.3733	0.2118	0.4149	0.4182
Fold 7	65.0	47.0	28.0	20.0	0.7000	0.3000	0.7647	0.6989	0.7303	0.3733	0.2353	0.3914	0.3938
Fold 8	64.0	48.0	27.0	21.0	0.7000	0.3000	0.7529	0.7033	0.7273	0.3600	0.2471	0.3929	0.3948
Fold 9	65.0	55.0	20.0	20.0	0.7500	0.2500	0.7647	0.7647	0.7647	0.2667	0.2353	0.4980	0.4980
Fold 10	62.0	52.0	22.0	23.0	0.7170	0.2830	0.7294	0.7381	0.7337	0.2973	0.2706	0.4321	0.4317
Average	67.3	50.0	24.4	18.2	0.7336	0.2664	0.7870	0.7342	0.7592	0.3279	0.2130	0.4591	0.4617



📊 Summary for GRU

	TP	TN	FP	FN	Accuracy	Error Rate	Recall	Precision	F1 Score	FPR	FNR	TSS	HSS
Fold 1	66.0	56.0	18.0	20.0	0.7625	0.2375	0.7674	0.7857	0.7765	0.2432	0.2326	0.5242	0.5232
Fold 2	64.0	54.0	20.0	22.0	0.7375	0.2625	0.7442	0.7619	0.7529	0.2703	0.2558	0.4739	0.4730
Fold 3	73.0	49.0	25.0	13.0	0.7625	0.2375	0.8488	0.7449	0.7935	0.3378	0.1512	0.5110	0.5168
Fold 4	64.0	56.0	18.0	22.0	0.7500	0.2500	0.7442	0.7805	0.7619	0.2432	0.2558	0.5009	0.4991
Fold 5	63.0	58.0	16.0	23.0	0.7562	0.2438	0.7326	0.7975	0.7636	0.2162	0.2674	0.5163	0.5130
Fold 6	66.0	52.0	23.0	19.0	0.7375	0.2625	0.7765	0.7416	0.7586	0.3067	0.2235	0.4698	0.4713
Fold 7	62.0	55.0	20.0	23.0	0.7312	0.2688	0.7294	0.7561	0.7425	0.2667	0.2706	0.4627	0.4617
Fold 8	66.0	56.0	19.0	19.0	0.7625	0.2375	0.7765	0.7765	0.7765	0.2533	0.2235	0.5231	0.5231
Fold 9	60.0	59.0	16.0	25.0	0.7438	0.2562	0.7059	0.7895	0.7453	0.2133	0.2941	0.4925	0.4891
Fold 10	57.0	64.0	10.0	28.0	0.7610	0.2390	0.6706	0.8507	0.7500	0.1351	0.3294	0.5355	0.5272
Average	64.1	55.9	18.5	21.4	0.7505	0.2495	0.7496	0.7785	0.7621	0.2486	0.2504	0.5010	0.4997

9. Project ZIP Contents

- kollikonda_dathwik_finalproject.ipynb

- kollikonda_dathwik_finalproject.py

- winequality-red.csv

- This report

10. Resources and References

https://www.datacamp.com/tutorial/random-forests-classifier-python

https://archive.ics.uci.edu/ml/index.php

https://scikit-learn.org/stable/

https://archive.ics.uci.edu/dataset/186/wine+quality

https://www.tensorflow.org/