Predicting RED Wine Quality Using K-Nearest Neighbours

Team Name : CRACK CODE

Tarun Kumar(2025JRB2025)

KNN Regressor Analysis

1. Introduction

This report presents an analytical study on the Red Wine Quality dataset using three regression models:

- **KNN Regressor (Custom Implementation)**
- **Linear Regression**
- **Ridge Regression**

The goal is to predict the wine quality score (0–10 scale) based on 11 physico-chemical features.

2. Data Preprocessing

The dataset originally contained **1599 rows**. After cleaning:

- **240 duplicated rows** were removed (final size: 1359 rows)
- No missing values were found
- A **70:30 train-test split** was applied
- **PowerTransformer (Yeo–Johnson)** was used to normalize the data

Why PowerTransformer?

It performs variance stabilization, handles skewness, and applies internal standardization, improving model performance.

3. Models Implemented

3.1. 1. Custom KNN Regressor

- Distance metric: **Euclidean/manhattan/minkowski**
- Weighting: **Distance/uniform**
- Neighbors: **k = 14 (you will take other values also)**

3.2. 2. Linear Regression

A standard Ordinary Least Squares (OLS) model to test linear relationships.

3.3. 3. Ridge Regression

- Regularization parameter: $\alpha = 12.5$
- Helps prevent overfitting by penalizing large coefficients

4. Evaluation Metrics

All models were evaluated using:

- **Mean Squared Error (MSE)**
- **Root Mean Squared Error (RMSE)**
- R^2 **Score**
- **5-Fold Cross-Validation Score**

5. Results: 70:30 Train-Test Split

Table 1: **Model Performance (70:30 Split)**

Model	R^2	MSE	RMSE	CV Score
KNN Regressor	0.4003	0.4059	0.6371	0.3048
Linear Regression	0.3826	0.4178	0.6464	0.3272
Ridge Regression	0.3834	0.4173	0.6460	0.3280

6. Additional Experiment: 80:20 Split and Cross-Validation

To evaluate the stability of the KNN model, a separate **80:20 train-test split** experiment was performed.

KNN Regressor Results (80:20 Split)

R^2 Score: 0.39603233101353963

5-Fold CV Scores:

$$[0.35916143,\ 0.34499895,\ 0.30787359,\ 0.2388267,\ 0.34455723]$$

Mean CV R^2: 0.31908357987510005

These results show that KNN performs consistently across different split ratios.

7. Hypertuning of *k*

To optimize the KNN Regressor, a range of values for the number of neighbors was tested using **GridSearchCV**. The search space used was:

$$k \in [1, 5, 9, 13, 17, 21, 25, 29, 33, 37]$$

Optimal k from GridSearchCV

- **Best k:** 25
- **Best Score (5-fold CV):** 0.3282701283464921

The hypertuning results indicate that performance improved when k was increased from the default value (14) to 25, demonstrating the importance of hyperparameter tuning in KNN-based models.

8. Comparative Analysis

- **KNN achieved the best test-set R^2 (0.4003)** among the models.
- **Ridge Regression showed the best cross-validation stability**.
- KNN obtained the **lowest MSE and RMSE**, indicating stronger prediction performance.
- Linear models struggled due to the nonlinear nature of relationships in the dataset.

9. Conclusion

- **Best test-set model:** KNN Regressor
- **Best generalization:** Ridge Regression
- Power-transformed features significantly improved model stability.

Future improvements:

- Try ensemble models (Random Forest, XGBoost)
- Perform deeper hyperparameter tuning
- Explore polynomial feature expansion

KNN Classification Analysis

1. Introduction

This report presents a detailed classification study on the Red Wine Quality dataset. The goal is to classify wine samples as:

$$\textbf{Good Quality (1) if quality} \geq 7, \quad \textbf{else Poor Quality (0)}$$

Five classification models were implemented:

- **Custom KNN Classifier (from scratch)**
- **Logistic Regression**
- **Decision Tree Classifier**
- **Random Forest Classifier**
- **Support Vector Classifier (SVC)**

All results below are extracted from the provided PDF. :contentReference[oaicite:1]index=1

2. Data Preprocessing

The dataset preprocessing steps:

- Original size: **1599 rows × 12 columns**
- **240 duplicated rows** removed $\rightarrow$ final size: **1359 rows**
- Converted to binary classification using `Quality_Label`
- Features standardized using **PowerTransformer (Yeo–Johnson)**
- Train-test split: **80:20** (1087 train, 272 test)

Why PowerTransformer?

The Yeo–Johnson PowerTransformer removes skewness, stabilizes variance, and improves classifier performance.

3. Custom KNN Classification

A KNN classifier was implemented entirely from scratch.

Performance of Custom KNN

KNN (No PCA)

- **Accuracy:** 0.9007
- **Precision:** 0.6296
- **Recall:** 0.50
- **F1 Score:** 0.5573
- **Cross-Validation Mean Score:** 0.8666
- **ROC-AUC:** 0.8150

KNN with PCA

PCA Components Used: 8

KNN (with PCA)

Classification Accuracy: 88.60%

4. Comparison with Standard Classifiers

4.1. 1. Logistic Regression

Accuracy: 0.9080
Precision: 0.6666
Recall: 0.5294
F1 Score: 0.5901
CV Score: 0.8684
ROC-AUC: 0.8914

4.2. 2. Decision Tree Classifier

Accuracy: 0.8860
Precision: 0.5405
Recall: 0.5882
F1 Score: 0.5633
CV Score: 0.8288
ROC-AUC: 0.7584

4.3. 3. Random Forest Classifier

Accuracy: 0.9007
Precision: 0.6842
Recall: 0.3823
F1 Score: 0.4905
CV Score: 0.8674
ROC-AUC: 0.8967

4.4. 4. Support Vector Classifier (SVC)

Accuracy: 0.9080
Precision: 0.8000
Recall: 0.3529
F1 Score: 0.4898
CV Score: 0.8620
ROC-AUC: 0.8732

5. Handling Class Imbalance (SMOTE + ENN)

Dataset imbalance:

$$0 : 86.4\%, \qquad 1 : 13.5\%$$

After SMOTE+ENN:

$$0 : 55.2\%, \qquad 1 : 44.7\%$$

KNN After Balancing

KNN After SMOTE+ENN

Accuracy: 0.7169
Precision: 0.2828
Recall: 0.8235
F1 Score: 0.4210
CV Score (Pipeline): 0.6927

6. Hyperparameter Tuning (GridSearchCV)

6.1. 1. KNN Classifier

Best Params: $k = 9,\ weights = uniform$

Best Accuracy: 0.8693

6.2. 2. Logistic Regression

Best Params: $C = 0.1,\ max\_iter = 50,\ class\_weight = None$

Best Score: 0.8693

6.3. 3. Decision Tree

Best Params: $criterion = entropy,\ max\_depth = 5,\ max\_features = 1.0$

Best Score: 0.8611

6.4. 4. Random Forest

Best Params: $n\_estimators = 200,\ max\_depth = 10,\ max\_samples = 0.25$

Best Score: 0.8785

6.5. 5. SVC

Best Params: $C = 1,\ kernel = rbf,\ degree = 2,\ gamma = scale$

Best Score: 0.8711

7. Comparative Analysis

This section summarizes a direct comparison of all five classification models based on accuracy, precision, recall, F1-score, cross-validation performance, and ROC-AUC.

Overall Performance Comparison

Accuracy Comparison

- Logistic Regression: **0.9080**
- SVC: **0.9080**
- Random Forest: 0.9007
- Custom KNN: 0.9007
- Decision Tree: 0.8860

ROC-AUC Comparison

ROC-AUC Scores

- Logistic Regression: **0.8914**
- Random Forest: **0.8967** (highest)
- SVC: 0.8732
- Custom KNN: 0.8150
- Decision Tree: 0.7584

Model Behavior Insights

- **Logistic Regression and SVC** achieved the highest accuracy, showing that linear and kernel-based decision boundaries work well on this dataset.
- **Random Forest** demonstrated the best ROC-AUC, indicating superior ranking and probability estimation performance.
- **KNN performed competitively** but showed sensitivity to data distribution and class imbalance.
- **Decision Tree** underperformed slightly due to high variance and overfitting risk.
- **SMOTE+ENN improved recall significantly** for the minority class but reduced accuracy for KNN.

Best Models

Final Model Ranking

1. **Logistic Regression** — Best balanced accuracy + stable CV
2. **SVC** — Best when precision is prioritized
3. **Random Forest** — Best ROC-AUC (ranking ability)
4. **KNN** — Strong but sensitive to imbalance
5. **Decision Tree** — Fast but lowest generalization

This comparison shows that **Logistic Regression and SVC** are the top-performing classifiers for this dataset, while Random Forest provides the strongest probabilistic discrimination.

8. Conclusion

Overall Findings

- Logistic Regression and SVC achieved the highest accuracy (0.9080)
- Random Forest achieved the best ROC-AUC after Logistic Regression
- KNN performed well but improved with PCA
- SMOTE+ENN greatly increased recall for minority class
- Random Forest had the best hyperparameter-tuned accuracy (0.8785)

Future directions include:

- Ensemble stacking
- XGBoost / LightGBM classifiers
- Better sampling techniques like ADASYN