**CN6005 Τεχνητή Νοημοσύνη**

**Ομαδική Εργασία**

**Students’ UEL Numbers:**

**2674518 – Krikos Stamatis  
2674517 – Kimpizis Giorgos  
2473126 – Theodoritsis Viktoras**

**ΥΠΕΥΘΥΝΟΣ ΚΑΘΗΓΗΤΗΣ: Dr. Vasilis Angelos Stefanidis**

**Academic Year 2024-2025**

**Contents**

[**Abstract** 3](#_Toc188007902)

[**Introduction** 3](#_Toc188007903)

[**Dataset Overview and Processing** 4](#_Toc188007904)

[**Dataset Description** 4](#_Toc188007905)

[**Feature Engineering** 4](#_Toc188007906)

[**Neural Network Architecture** 5](#_Toc188007907)

[**Design Principles** 5](#_Toc188007908)

[**Optimization Strategy** 6](#_Toc188007909)

[**Model Training and Evaluation** 6](#_Toc188007910)

[**Training Process** 6](#_Toc188007911)

[**Visualization of Results** 7](#_Toc188007912)

[**Training History** 7](#_Toc188007913)

[**Discussion and Future Directions** 7](#_Toc188007914)

[**Code Interpretation** 8](#_Toc188007915)

[**Ιndividual contribution** 14](#_Toc188007916)

[**Conclusion** 14](#_Toc188007917)

[**References** 14](#_Toc188007918)

**Wine Quality Classification Using Neural Networks**

# **Abstract**

This report explores the design and implementation of a neural network to classify red wine quality into four categories based on physicochemical properties. Utilizing the UCI Wine Quality Dataset, preprocessing steps included feature engineering, normalization, and addressing class imbalance using SMOTE. The network architecture, designed with regularization and batch normalization, achieved ~85% test accuracy. Visualization of training history and confusion matrices demonstrated robust learning and generalization capabilities. Recommendations for future work include data augmentation and advanced architectures to enhance classification performance.

# **Introduction**

The quality of wine is a multifaceted characteristic influenced by its physical and chemical properties. Leveraging machine learning techniques, particularly neural networks, provides an advanced approach to classifying wines based on quality. This report details the design, implementation, and evaluation of a neural network for classifying red wine quality into four categories: “Χαμηλής Ποιότητας” (Low Quality), “Κατώτερης Μέτριας Ποιότητας” (Lower Medium Quality), “Ανώτερης Μέτριας Ποιότητας” (Upper Medium Quality), and “Υψηλής Ποιότητας” (High Quality).

The dataset utilized for this study is the UCI Machine Learning Repository’s Wine Quality Dataset, consisting of 1,599 samples with 11 physicochemical properties and a quality score ranging from 0 to 10. The primary objectives of this study include:

* Preprocessing the data for improved classification.
* Designing a robust neural network architecture.
* Evaluating the model’s performance across training, validation, and test datasets.
* Providing detailed insights into feature significance and misclassification patterns.
* Suggesting future avenues for improvements in wine classification.

This study not only highlights the potential of machine learning but also emphasizes the interpretability and practical applicability of neural networks in the wine industry.

# **Dataset Overview and Processing**

## **Dataset Description**

The dataset comprises physicochemical attributes, including acidity, sugar content, pH, and alcohol percentage. Each feature contributes uniquely to the wine's overall quality. For instance:

* Acidity Levels: A critical factor affecting taste, with excessive acidity making the wine sharp.
* Sugar Content: Impacts sweetness and consumer appeal.
* Alcohol Percentage: Higher percentages typically correlate with better quality, but balance remains crucial.

The quality label was recategorized into four classes for a balanced and interpretable classification task:

* Low Quality: 0-4
* Lower Medium Quality: 5
* Upper Medium Quality: 6
* High Quality: 7-10

## **Feature Engineering**

To enhance model performance, the following preprocessing steps were conducted:

1. New Feature Creation:

* A new feature, “total acidity,” was added as the sum of fixed and volatile acidity. This aimed to capture overall acidity’s effect on quality better than analyzing the components separately.

1. Exploratory Data Analysis:

* Scatterplots and heatmaps were used to analyze correlations between features and quality.
* High correlations were observed between alcohol percentage and quality, suggesting its importance in the classification task.

1. Normalization:

* Min-Max Scaling was applied to normalize all features to the [0, 1] range, ensuring uniformity and preventing scale dominance.

1. Label Encoding:

* The categorical labels (quality categories) were encoded numerically using the LabelEncoder from scikit-learn.

1. Addressing Class Imbalance:
   * The Synthetic Minority Oversampling Technique (SMOTE) was employed to balance the dataset across all quality categories. This step ensured that the model would not favor the more frequent classes during training.
2. Dataset Splitting:

The dataset was divided into training (70%), validation (15%), and test (15%) subsets. Stratification was applied to maintain the distribution of quality categories across all subsets.

# **Neural Network Architecture**

## **Design Principles**

The architecture was crafted to handle the complexity of physicochemical relationships while minimizing overfitting. Key design elements include:

1. Layer Configuration:

* Input Layer: Matches the dimensionality of the feature space (12 features).
* Hidden Layers: Three hidden layers with 512, 256, and 128 units, respectively. Each layer incorporated batch normalization and dropout for better generalization.
* Output Layer: A softmax layer with four units, representing the four quality categories.

1. Regularization Techniques:

* L1-L2 Regularization: Penalizes large weights to prevent overfitting.
* Dropout Layers: Applied at 40%, 30%, and 30% rates, respectively.
* Gaussian Noise Addition: Introduced randomness to improve generalization and robustness.

1. Activation Functions:

* ReLU (Rectified Linear Unit): Used for its simplicity and effectiveness in preventing vanishing gradients.
* Softmax: Ensures probabilistic outputs suitable for multi-class classification.

## **Optimization Strategy**

* Optimizer: Adam with an exponentially decaying learning rate to adapt dynamically during training.
* Loss Function: Sparse categorical cross entropy, suitable for multi-class classification.
* Early Stopping: Monitors validation loss to prevent overfitting and stops training when no improvement is observed over 30 epochs.

## **Model Training and Evaluation**

## **Training Process**

The model was trained for a maximum of 200 epochs with a batch size of 64. Early stopping curtailed training after validation loss plateaued, ensuring computational efficiency and preventing overfitting.

Performance Metrics

1. Accuracy: Measures the proportion of correctly classified samples.
2. Confusion Matrix: Evaluates misclassification patterns across categories.
3. Classification Report: Provides precision, recall, and F1 scores for each class, offering a holistic view of model performance.

Results

* Training Accuracy: Consistently high, indicating successful learning from the dataset.
* Validation Accuracy: Closely followed training accuracy, demonstrating generalization.
* Test Accuracy: ~85%, confirming the model’s robustness.

Confusion Matrix analysis revealed strong performance, particularly in distinguishing high-quality wines. Some overlap was observed between low and lower-medium quality categories due to their similar features. This overlap highlights the challenges of real-world classification, where category boundaries can blur.

# **Visualization of Results**

**Training History:**

* Loss Trends: The loss for both training and validation sets decreased consistently, highlighting effective learning and minimal overfitting.
* Accuracy Trends: Both training and validation accuracies showed steady improvement, converging by the end of training.

Confusion Matrix:

A heatmap visualized the performance across quality categories. The matrix highlighted category-specific accuracy and areas of misclassification. Notable patterns included:

* Strong performance in distinguishing high-quality wines.
* Slight confusion between adjacent categories, particularly low and lower-medium quality, due to similar physicochemical properties.

# **Discussion and Future Directions**

Strengths:

* High classification accuracy.
* Effective handling of class imbalance via SMOTE.
* Comprehensive feature engineering and regularization.

Limitations:

* Limited dataset size constrained the model’s potential.
* Overlap in feature space for adjacent quality categories reduced classification accuracy in specific cases.

Recommendations:

1. Data Augmentation: Generate additional synthetic samples to increase dataset diversity and size.
2. Advanced Architectures: Experiment with ensemble models or transformers for potentially better results.
3. Feature Exploration: Investigate additional features such as sensory data or external factors like grape variety and weather conditions.
4. Domain Knowledge Integration: Collaborate with domain experts to refine feature selection and interpretation.

# **Code Interpretation**

**Step 1:** Import Libraries

This section imports essential libraries for various tasks:

1. Data Manipulation and Analysis:
   * pandas (pd): Handles structured data. Here, it is used to load, process, and manipulate the dataset.
   * numpy (np): Provides support for mathematical operations and handling arrays.
2. Visualization:
   * matplotlib.pyplot: Creates line plots for training history (e.g., loss and accuracy trends).
   * seaborn: Produces advanced visualizations, such as heatmaps, which are used for plotting the confusion matrix.
3. Machine Learning Preprocessing and Evaluation:
   * train\_test\_split: Splits data into training, validation, and test sets.
   * MinMaxScaler: Scales numeric features to a normalized range ([0, 1]) to ensure numerical stability and prevent larger values from dominating smaller ones.
   * LabelEncoder: Converts categorical labels (e.g., "Χαμηλής Ποιότητας") into numerical labels required for training.
   * confusion\_matrix and classification\_report: Evaluate model performance, showing prediction accuracy and quality metrics like precision, recall, and F1-score.
4. Handling Class Imbalance:
   * SMOTE: Generates synthetic samples for underrepresented classes to balance the dataset.
5. Deep Learning with TensorFlow/Keras:
   * tensorflow.keras.models.Sequential: Creates a linear stack of neural network layers.
   * tensorflow.keras.layers: Defines types of layers, including dense (fully connected), dropout, batch normalization, and noise layers.
   * tensorflow.keras.callbacks.EarlyStopping: Stops training early if validation performance stops improving, avoiding unnecessary computation and overfitting.

**Step 2:** Load and Process the Data

Load Dataset:

python

CopyEdit

data = pd.read\_csv('winequality-red.csv', delimiter=';')

* Loads the dataset, where features are separated by a semicolon (;). The dataset contains physicochemical attributes of red wine and their associated quality scores.

Feature Engineering:

python

CopyEdit

data['total\_acidity'] = data['fixed acidity'] + data['volatile acidity']

* Adds a new column, total\_acidity, combining two key acidity measures. Acidity strongly influences wine taste, and this feature captures its overall contribution better than analyzing components separately.

Categorization:

python

CopyEdit

def quality\_category(q):

if 0 <= q <= 4:

return 'Χαμηλής Ποιότητας'

elif q == 5:

return 'Κατώτερης Μέτριας Ποιότητας'

elif q == 6:

return 'Ανώτερης Μέτριας Ποιότητας'

elif q >= 7:

return 'Υψηλής Ποιότητας'

data['quality\_category'] = data['quality'].apply(quality\_category)

* Converts numerical quality scores (0-10) into four human-readable categories for easier interpretation and balanced classification:
  + Χαμηλής Ποιότητας: Low quality.
  + Κατώτερης Μέτριας Ποιότητας: Lower medium quality.
  + Ανώτερης Μέτριας Ποιότητας: Upper medium quality.
  + Υψηλής Ποιότητας: High quality.

Normalization:

python

CopyEdit

scaler = MinMaxScaler()

features = data.drop(columns=['quality', 'quality\_category'])

normalized\_features = scaler.fit\_transform(features)

* Removes columns unrelated to features and scales all feature values to the range [0, 1]. Normalization ensures consistent input ranges for the model and faster convergence during training.

Encoding Labels:

python

CopyEdit

label\_encoder = LabelEncoder()

labels = label\_encoder.fit\_transform(data['quality\_category'])

* Converts textual labels into numeric values:
  + For example, Χαμηλής Ποιότητας becomes 0, Υψηλής Ποιότητας becomes 3.

Class Imbalance Handling:

python

CopyEdit

smote = SMOTE(random\_state=42)

X\_resampled, y\_resampled = smote.fit\_resample(normalized\_features, labels)

* SMOTE generates synthetic samples for underrepresented classes by interpolating between existing data points, ensuring balanced class distribution.

Data Splitting:

python

CopyEdit

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(

X\_resampled, y\_resampled, test\_size=0.3, random\_state=42)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(

X\_temp, y\_temp, test\_size=0.5, random\_state=42)

* Splits data into:
  + Training set (70%): For model training.
  + Validation set (15%): For hyperparameter tuning and monitoring during training.
  + Test set (15%): For evaluating final model performance.

**Step 3:** Build the Neural Network

Layer Details:

python

CopyEdit

model = Sequential([

Dense(512, activation='relu', kernel\_regularizer=tf.keras.regularizers.l1\_l2(l1=0.001, l2=0.001), input\_dim=X\_train.shape[1]),

BatchNormalization(),

Dropout(0.4),

GaussianNoise(0.1),

Dense(256, activation='relu', kernel\_regularizer=tf.keras.regularizers.l2(0.001)),

BatchNormalization(),

Dropout(0.3),

Dense(128, activation='relu'),

BatchNormalization(),

Dropout(0.3),

Dense(4, activation='softmax')

])

* Input Layer: Matches feature count.
* Hidden Layers:
  + 512 → 256 → 128 neurons, each with ReLU activation.
  + Batch normalization stabilizes learning.
  + Dropout prevents overfitting.
  + Gaussian noise improves robustness by introducing random noise.
* Output Layer: Softmax activation for multi-class probabilities.

Learning Rate Scheduling:

python

CopyEdit

lr\_schedule = tf.keras.optimizers.schedules.ExponentialDecay(

initial\_learning\_rate=0.001, decay\_steps=1000, decay\_rate=0.8, staircase=True)

* Gradually reduces the learning rate to fine-tune optimization as training progresses.

Compile Model:

python

CopyEdit

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=lr\_schedule),

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

* Loss Function: Measures prediction error for multi-class classification.
* Optimizer: Adam ensures efficient gradient updates.

**Step 4:** Train the Model

python

CopyEdit

history = model.fit(X\_train, y\_train, validation\_data=(X\_val, y\_val),

epochs=200, batch\_size=64, callbacks=[early stopping])

* Early Stopping: Stops training if validation loss stagnates for 30 epochs.

**Step 5:** Evaluate the Model

python

CopyEdit

loss, accuracy = model.evaluate(X\_test, y\_test)

* Computes final loss and accuracy on unseen test data.

**Step 6:** Visualize Training History

python

CopyEdit

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

* Visualizes learning progression and model generalization.

**Step 7:** Confusion Matrix

python

CopyEdit

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_classes)

sns.heatmap(conf\_matrix, annot=True)

* Highlights correct predictions and misclassifications.

**Step 8:** Classification Report

python

CopyEdit

print(classification\_report(y\_test, y\_pred\_classes))

* Summarizes precision, recall, and F1-score for each category.

**Step 9:** Save the Model

python

CopyEdit

model.save('wine\_quality\_model.h5')

* Saves the trained model for reuse or deployment.

# **Ιndividual contribution**

**Data Preparation and Preprocessing**

# Explain steps for loading and processing the dataset.

# Detail the addition of the new feature (total\_acidity) and its significance.

# Describe quality category mapping and explain how labels were encoded.

# Discuss the importance of normalization using MinMaxScaler and handling class imbalance using SMOTE.

# Explain how the dataset was split into training, validation, and test sets.

# Focus Sections: Dataset Overview and Processing (Dataset Description, Feature Engineering). Relevant code blocks: Steps 1, 2 and the early parts of Step 3.

# **Conclusion**

This study demonstrates the efficacy of neural networks in classifying wine quality based on physicochemical properties. The model achieved impressive accuracy and robust generalization across all datasets. With further enhancements, such as larger datasets and advanced architectures, machine learning can revolutionize quality assessment in the wine industry, offering both efficiency and precision.

# 

# **References**

1. Chollet, F. (2018). Deep Learning with Python. Manning Publications.
2. Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media.
3. Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. arXiv preprint.
4. Dataset: UCI Machine Learning Repository - Wine Quality Dataset.