ESE417-Final Report

Group Members: Hang Yang, Xuyang Zheng, Siyuan He

Abstract: Red wine has become a common beverage in people's daily lives. Therefore, it is crucial to analyze red wine quality effectively before consumption. This project utilizes machine learning techniques to predict red wine quality based on various physicochemical properties. We used the red wine quality dataset from the UC Irvine Machine Learning Repository and applied three classification algorithms: Random Forest, Support Vector Machine, and Artificial Neural Network. Through calculating multiple performance metrics and comparing results between training and test sets, our experiments demonstrate that Random Forest performs best in the red wine quality classification task.

I. Introduction

Red wine, an alcoholic beverage derived from the fermentation of purple or black grapes, is widely utilized in social contexts. Multiple variables, including grape varietal, climatic conditions, soil composition, and vinification processes, determine the comprehensive quality of wine. These parameters contribute to substantial organoleptic properties and quality variations, presenting significant challenges for consumers in quality wine selection.

Machine learning techniques, leveraging wine's physicochemical characteristics, provide robust tools for quality assessment. Cortez et al. [1] previously introduced a methodology based on the "taste expectation framework," implementing Support Vector Machine, Naive Bayes, and Random Forest algorithms to analyze oenological engineering data. In this study, we employ supervised learning algorithms to classify red wine quality using 11 features extracted from the dataset.

We implemented comprehensive data preprocessing techniques to enhance model prediction accuracy and utilized Grid Search with 5-fold cross-validation to optimize model weight factors and hyperparameters. The final model achieved 91.10% accuracy through multiple iterations and experimental validation.

II. Exploratory Analysis of the Dataset

This investigation utilizes data from the UCI Machine Learning Repository [2], encompassing 1,599 instances, each characterized by 12 variables that represent red wine's physicochemical properties and quality assessments. The research methodology employs these input features to develop predictive red wine quality evaluation models.

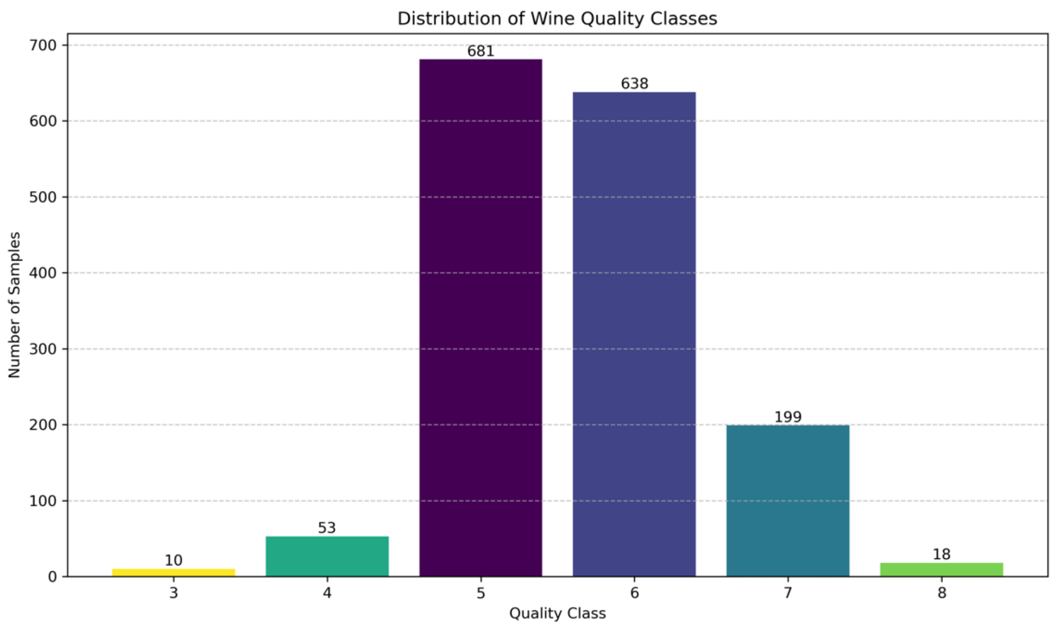


Fig. 1. Quality distribution of the dataset

As illustrated in Figure 1, the quality ratings exhibit a distribution ranging from 3 to 8, with "3" denoting inferior quality and "8" signifying superior quality. The distribution demonstrates a central tendency, with most samples clustered in the 5-6 rating range. In the context of machine learning evaluation metrics, confusion matrices serve as primary tools for visualizing classification model performance. We implemented a Pearson correlation coefficient matrix to facilitate comprehensive feature relationship analysis and identify underlying correlations (see Figure 2). The matrix elucidates the interrelationships among wine's physicochemical attributes, including alcohol content, volatile acidity, sulphates, citric acid, total sulfur dioxide, density, chlorides, fixed acidity, pH levels, free sulfur dioxide, and residual sugar.

The correlation heatmap in Figure 2 reveals significant correlations between wine quality and several physicochemical properties, specifically alcohol content, volatile acidity, citric acid, and sulphates. Furthermore, the heatmap identifies the following feature relationships:

1. Alcohol exhibits a weak positive correlation with pH values.
2. Citric acid demonstrates strong positive correlations with fixed acidity and density measurements.
3. pH values show inverse correlations with fixed acidity, citric acid, density, and sulphate levels.

To facilitate a deeper analysis of individual variable impacts on wine quality, we employed violin plot visualizations to illustrate the distribution patterns of key variables across different quality classifications (see Figures 3-6).

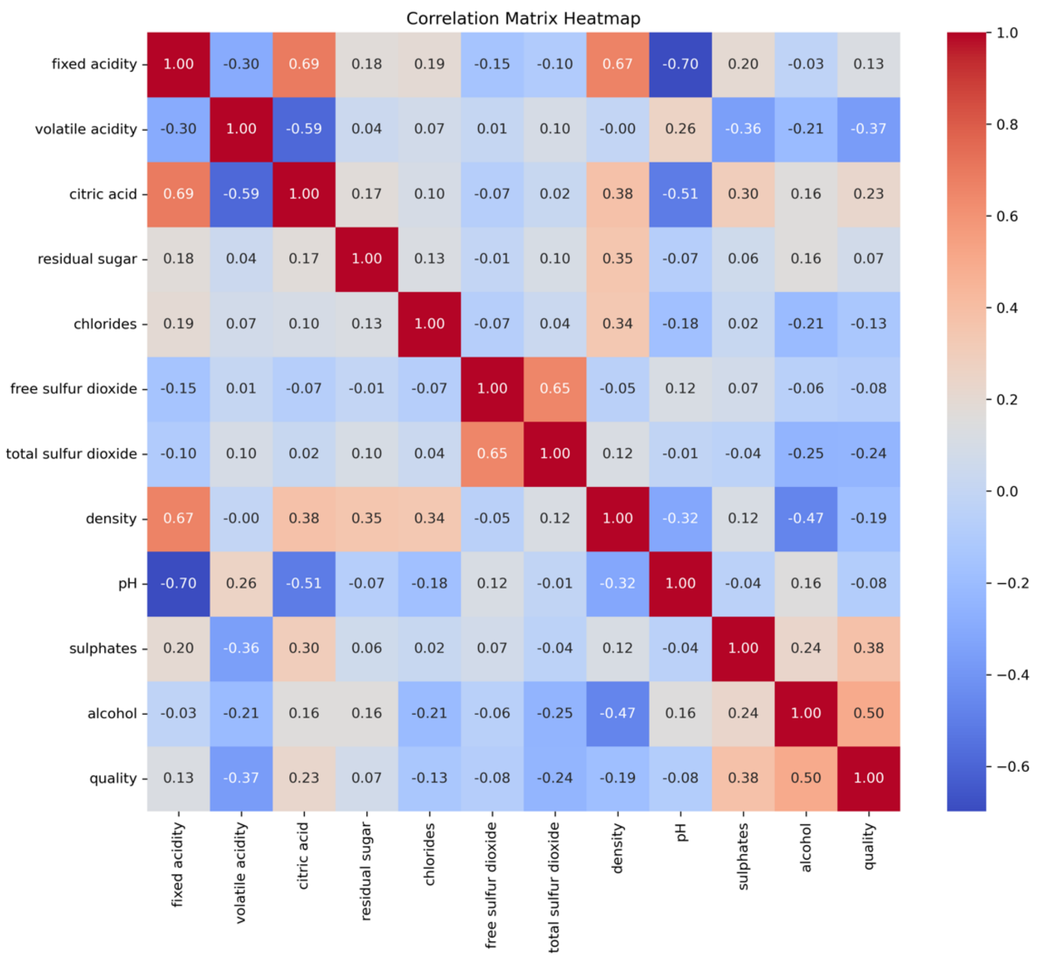


Fig. 2. Correlation Heatmap of Wine Quality Features

Figure 3 illustrates that fixed acidity distributions exhibit similar patterns across quality classifications, corroborating the weak correlation coefficient observed in the heatmap. Figure 4 reveals a positive correlation between alcohol content and quality grades, with particularly elevated alcohol concentrations in premium wines rated 7 and 8. Figure 6 demonstrates an inverse relationship between wine quality and volatile acidity levels. These observations are congruent with the correlation patterns depicted in the heatmap, providing additional validation for our analytical approach.

This exploratory data analysis has facilitated a comprehensive understanding of the dataset characteristics and their relationships with wine quality, establishing crucial groundwork for subsequent model development and parameter optimization.

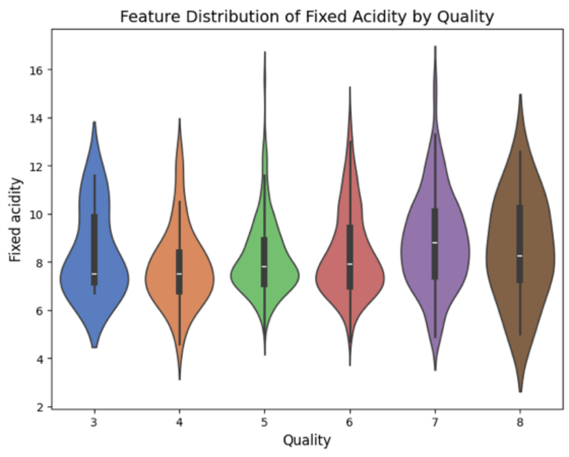


Fig. 3. Fixed Acidity Across Wine Quality Levels

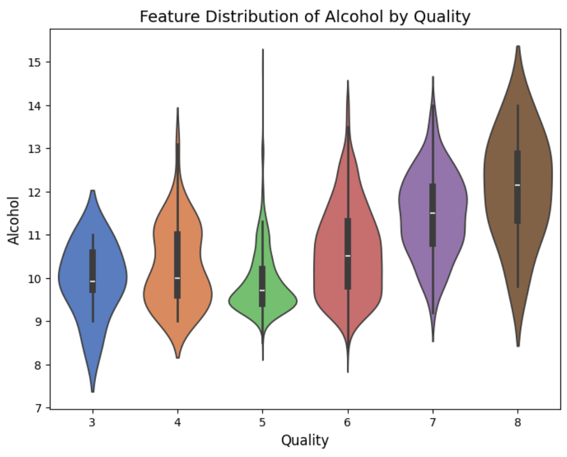


Fig. 4. Alcohol Content Across Wine Quality Levels

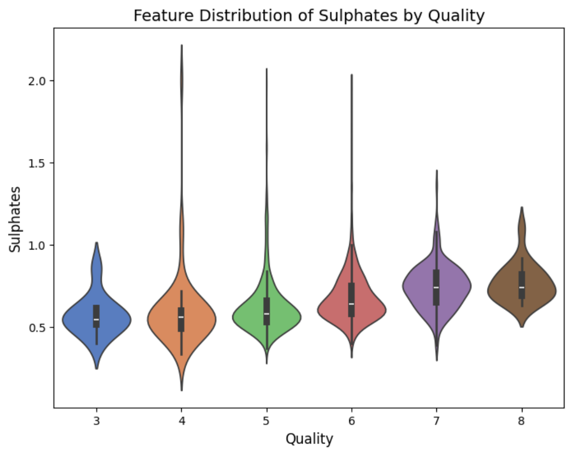


Fig. 5. Sulphates Across Wine Quality Levels

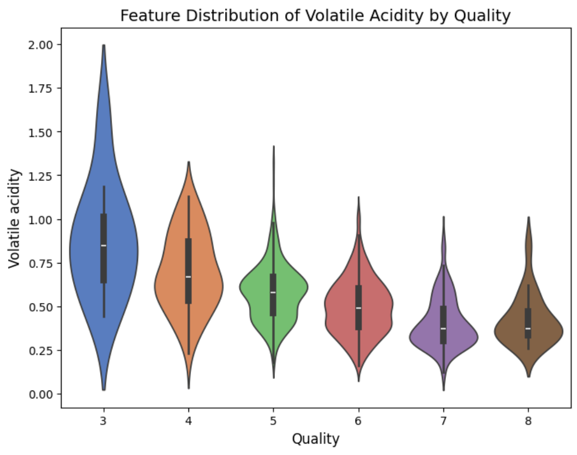


Fig. 6. Volatile Acidity Across Wine Quality Levels

A. Data Cleaning

Data cleaning is a crucial step in ensuring model performance. In this project, we first removed duplicate values and outliers from the dataset; for missing values, we employed mean, median, or mode imputation; to eliminate the impact of varying feature value ranges on the model, we applied Z-score normalization; finally, we split the data into training and test sets, preparing for model training and evaluation.

B. Feature Engineering

Feature engineering represents a critical component in optimizing model predictive capacity. The process begins with the computation of the dataset's correlation matrix, then extracting features exhibiting high correlation with the target variable (|correlation| > 0.6) as candidate features. For features demonstrating high intercorrelations, we explore various Principal Component Analysis (PCA) approaches for dimensionality reduction. For instance, given that feature A exhibits strong correlations with features B and C, we can implement either pairwise PCA (A with B, A with C) to generate two transformed feature vectors or simultaneous PCA across A, B, and C to produce a single composite feature vector. Subsequent experimental analysis will investigate the impact of these different PCA implementations on model accuracy across our three algorithms. PCA dimensionality reduction effectively addresses multicollinearity and the curse of dimensionality in high-dimensional feature spaces. In contrast, consolidating highly correlated features enables dimensional reduction while maximizing information retention.

C. Grid Search and Cross-validation

Grid Search represents a predominant methodology for hyperparameter optimization [3]. Its fundamental principle involves systematic traversal of user-defined hyperparameter spaces, facilitating comprehensive exploration of parameter combinations even within constrained search domains to identify optimal parameter configurations. Grid Search facilitates parallel computation due to the independence of individual parameter evaluations and their temporal non-interference. Furthermore, the outcome of each evaluation maintains independence from other iterations, ensuring computational flexibility. In this implementation, Grid Search was integrated with five-fold cross-validation, partitioning the dataset into five subsets, iteratively utilizing four subsets for training and 1 for validation, computing mean performance metrics across all potential parameter combinations to identify optimal configurations and subsequently retraining the model with the selected hyperparameters to maximize predictive performance. The hyperparameter optimization and selection process is illustrated in Figures 7-9.

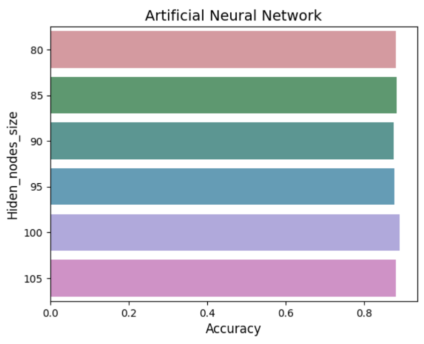


Fig. 7. Hidden Node Sizes on ANN Accuracy

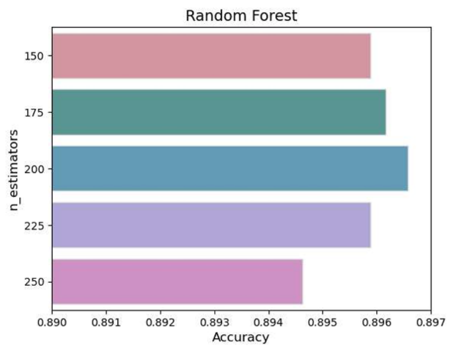


Fig. 8. Different Estimator Count on Random Forest Accuracy

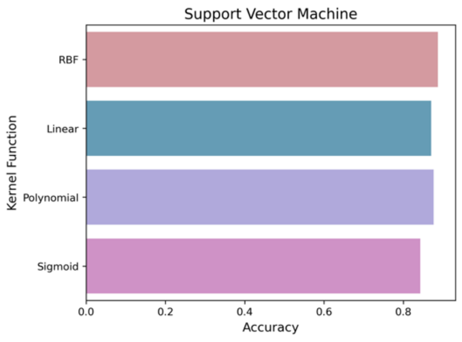


Fig. 9. Different Kernel Functions in SVM Accuracy

For Artificial Neural Network: The final configuration includes 100 nodes in the hidden layer and 2 nodes in the output layer. For Random Forest: The number of trees (estimators) is 200; max_depth is none, min_samples_split is 2. For Support Vector Machine: RBF kernel function is selected with parameters 'C': 95, 'gamma': 0.011, 'kernel': 'rbf.'

III. Method

In this project, we selected three machine learning methods: ANN, RF, and SVM. Below is a detailed introduction and implementation process for each method.

1. Artificial Neural Network

ANNs are generalizations simulating biological neural networks. ANNs consist of multiple layers of computational units connected through adjustable weights. Their main components include weights, biases, and activation functions. The choice of activation function is crucial for ANN performance, with common functions including Logistic function, ReLU, and SoftPlus [4]. The core concept of ANN is information transmission along predetermined paths between neurons, learning complex patterns through non-linear transformations. Its flexible architecture enhances performance through parameter adjustments.

In this project, we constructed an ANN model with multiple hidden layers, using optimized activation function and error function, updating weights through backpropagation. Model hyperparameters were tuned using Grid Search.

2. Random Forest

Random Forest is a decision tree method based on ensemble techniques, widely used in classification and regression tasks. This method generates the final output by constructing multiple decision trees and combining their predictions. Each tree relies on a random vector, ensuring diversity and robustness across the forest. Random Forest excels at handling high-dimensional data and non-linear relationships, effectively evaluating the importance of features for predicting key variables in classification and regression problems.

In this project, we built the Random Forest model using sklearn.ensemble.RandomForestClassifier module. The number of trees was set to 200; optimal combinations of maximum tree depth and minimum samples required to split nodes were selected through Grid Search and cross-validation.

3. Support Vector Machine

Introduced in the 1990s, the support vector machine (SVM) is a powerful machine learning algorithm widely used for classification and regression tasks. When used for regression, it's known as Support Vector Regression (SVR). SVR is a kernel-based technique that maps non-linear data from low-dimensional space to high-dimensional space through kernel tricks [5].

SVM's core concept is finding the optimal hyperplane in feature space to maximize the margin between classes. Kernel functions play a crucial role in SVM, with common kernels including linear, Sigmoid, Radial Basis Function (RBF), and polynomial.

In this project, we chose the RBF kernel for its superior performance on the wine dataset. The model was built using sklearn.svm.SVC module, with regularization parameter C and kernel coefficient Gamma optimized through Grid Search.

IV. Results and Analysis

We initially formulated the red wine quality prediction as a multiclass classification problem, categorizing samples based on their quality scores, with each discrete score representing a distinct quality level.

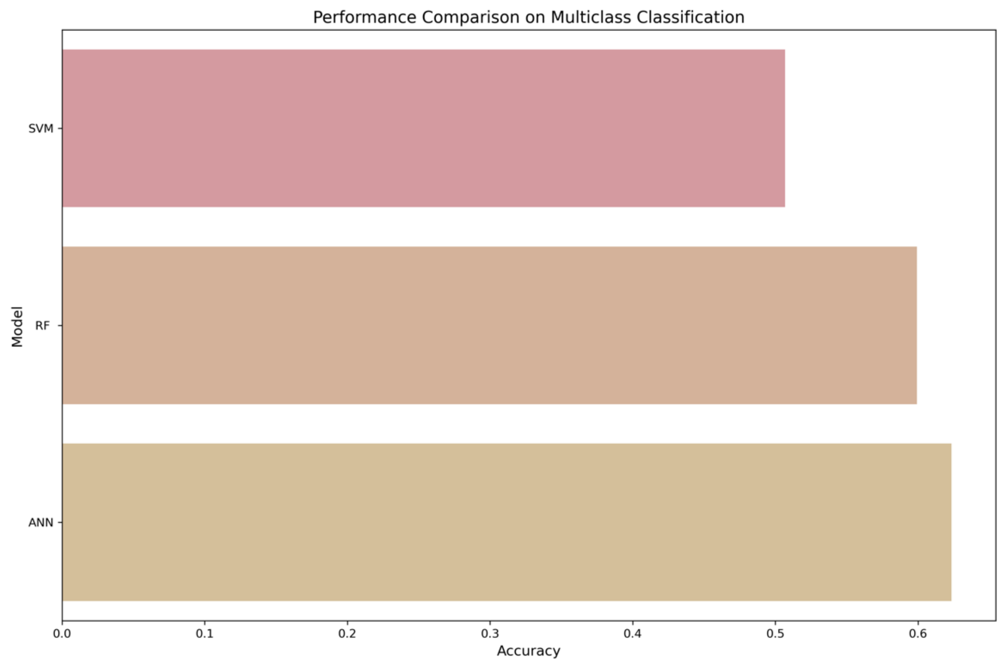


Fig. 10. Performance Comparison on Multiclass Classfication

As illustrated in Figure 10, the classification accuracy of the three implemented models - Support Vector Machine, Random Forest, and Artificial Neural Network. They all consistently fell within the 50%-65% range.

The observed suboptimal model performance can be primarily attributed to class imbalance in the dataset: the majority of samples cluster around quality scores 5 and 6, constituting the dominant classes, while samples with other quality ratings (3, 4, 7, 8) are underrepresented. This inhibits effective feature learning for these minority classes during model training and results in classification errors.

This class imbalance significantly compromised model classification performance. Consequently, we abandoned the direct multiclass classification approach in subsequent experiments and explored alternative strategies to enhance predictive accuracy.

Subsequently, we conducted comparative analyses of model performance across various data processing methodologies, including PCA dimensionality reduction to 5, 7, and 9 features, alongside the unprocessed dataset (no PCA). The experimental outcomes are presented in Figures 11-13.

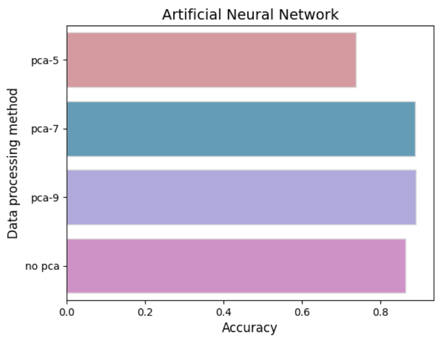


Fig. 11. Different PCA on ANN Accuracy

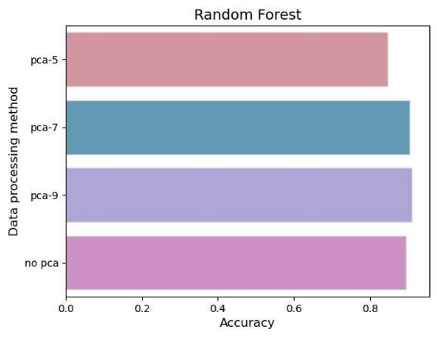


Fig. 12. Different PCA on RF Accuracy

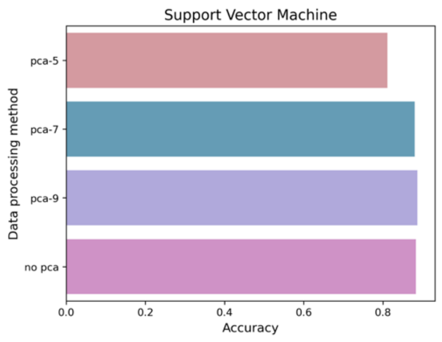


Fig. 13. Different PCA on SVM Accuracy

We implemented feature engineering through pairwise fusion of highly correlated features (fixed acidity with citric acid, fixed acidity with density, fixed acidity with pH, and free sulfur dioxide with total sulfur dioxide), progressively reducing feature dimensionality. PCA reduction to 9 features preserved maximum information content while eliminating two redundant features. Figures 11-13 demonstrate that all models achieved optimal performance with 9-dimensional PCA reduction, exhibiting peak accuracy rates. This empirically validates PCA-9 as the optimal dimensionality reduction strategy; further reduction to 5 features resulted in significant performance degradation due to critical information loss from excessive dimensionality reduction; while the unprocessed dataset (no PCA) offered operational simplicity, it demonstrated marginally inferior model precision compared to PCA-9 processed data.

The experimental evidence conclusively supports the selection of PCA-9 dimensionality reduction as the optimal data preprocessing strategy for maximizing model performance.

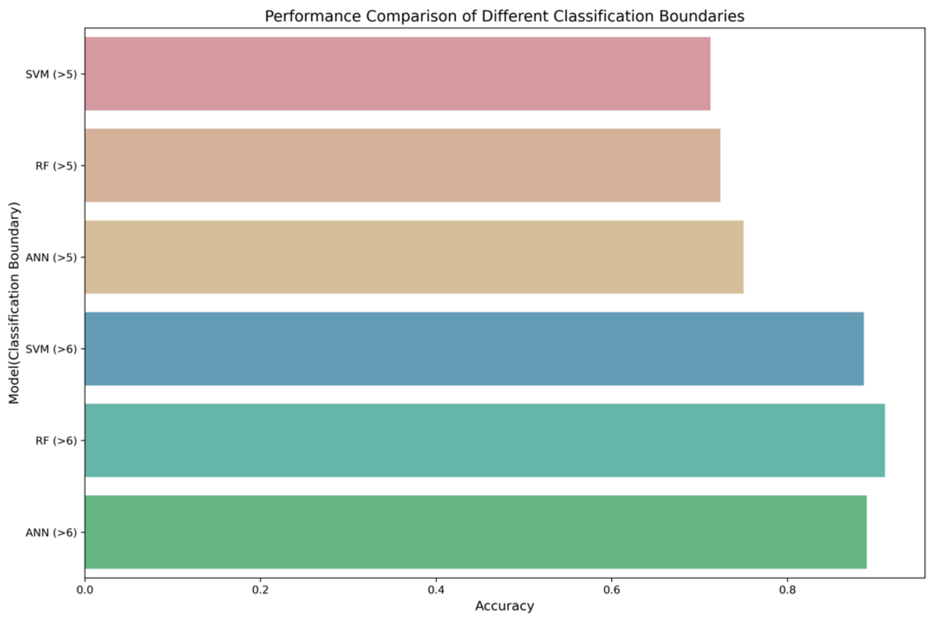


Fig. 14. Accuracy of Models with Different Classification Thresholds

After further optimizing the classification objectives, we redefined the red wine quality prediction task as binary classification problems, setting two classification boundaries: **quality > 5** and **quality > 6**. The experimental results are shown in Figure 14. From Figure 14, we can observe significant differences in model performance across different classification boundaries.

In the classification task of **quality > 5**, the lower overall accuracy can be attributed to the high similarity in features between quality levels 5 and 6. This resemblance made it challenging for the models to effectively distinguish between these two classes.

In the classification task of **quality > 6**, treating wines with quality levels 5 and 6 as the same class effectively reduced the class imbalance problem, significantly improving model classification performance.

Random Forest (RF) performed best in the **quality > 6** classification task, achieving an accuracy of **91.1%**, becoming the optimal classification model. Support Vector Machine (SVM) and Artificial Neural Network (ANN) also demonstrated stable performance, achieving good accuracy rates.

V. Conclusions

This study employed three mainstream machine learning classification algorithms—Random Forest (RF), Support Vector Machine (SVM), and Artificial Neural Network (ANN)—to conduct an in-depth analysis and prediction of the UCI red wine quality dataset. Through systematic data preprocessing, feature engineering, and hyperparameter optimization, we significantly enhanced the classification performance of the models.

In the dimensionality reduction analysis, the study systematically compared the results of PCA with different dimensions and found that reducing the features to 9 dimensions achieved the best performance, significantly outperforming both 5-dimensional PCA and the original dataset. This result indicates that reasonable dimensionality reduction can effectively improve model performance while retaining critical information and avoiding the adverse effects caused by data redundancy and high-dimensional features.

In the process of optimizing the classification strategy, we redefined the task as two binary classification problems: "quality > 5" and "quality > 6". This redefinition significantly improved the models' performance. Specifically, in the "quality > 6" task, merging quality levels 5 and 6 effectively alleviated the class imbalance problem, significantly improving the overall accuracy. In this classification task, Random Forest outperformed other methods, achieving an accuracy of 91.1%, making it the optimal classification solution. At the same time, Support Vector Machine and Artificial Neural Network also demonstrated good stability and high accuracy.

Throughout the experiment, the Random Forest algorithm exhibited significant advantages, with accuracy surpassing that of SVM and ANN.

VI. Contribution

Siyuan He worked on random forest, Xuyang Zheng worked on SVM, Hang Yang worked on ANN and data clearning. Report are equally distributed.

Reference

[1] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis,” Modeling wine preferences by data mining from physicochemical properties,” in IEEE Transactions on Neural Networks, vol. 17, no. 5, pp. 1136-1140, Sept. 2006. doi: 10.1109/TNN.2006.176

[2] Wine Quality-UCI Machine Learning Repository.

[3] Yu, Tong, and Hong Zhu. "Hyper-parameter optimization: A review of algorithms and applications." *arXiv preprint arXiv:2003.05689* (2020).

[4] Turian, J.P., Bergstra, J. and Bengio, Y. (2009) Quadratic Features and Deep Architectures for Chunking. Human Language Technologies Conference of the North American Chapter of the Association of Computational Linguistics, Boulder, Colorado, 31 May-5 June 2009, 245-248.

[5] Joshi, R.P., Eickholt, J., Li, L., Fornari, M., Barone, V. and Peralta, J.E. (2019) Machine Learning the Voltage of Electrode Materials in Metal-Ion Batteries. Journal of Applied Materials, 11, 18494-18503.
https://doi.org/10.1021/acsami.9b04933

December 16, 2024

```
[31]: import pandas as pd
      import numpy as np
      import matplotlib.pyplot as plt
      from collections import Counter
      from sklearn.metrics import accuracy_score
      import matplotlib.pyplot as plt
      from sklearn.model_selection import train_test_split
      from sklearn.preprocessing import MinMaxScaler, StandardScaler
      from sklearn.decomposition import PCA
```

```
[3]: # redwine dataset
     #
     #
     file_path = f'.\dataset\winequality-red-4.csv'
     with open(file_path, 'r') as f:
         column_line = f.readline().strip()
         column_names = column_line.split(";")

     #
     #
     #
     #
     data = pd.read_csv(file_path, skiprows=1, header=None, sep=";")
     cleaned_columns_name = [value.replace('"', '') for value in column_names]
     data.columns = cleaned_columns_name
```

```
[4]: data
```

```
[4]:       fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
     0               7.4             0.700         0.00             1.9      0.076
     1               7.8             0.880         0.00             2.6      0.098
     2               7.8             0.760         0.04             2.3      0.092
     3              11.2             0.280         0.56             1.9      0.075
     4               7.4             0.700         0.00             1.9      0.076
     …                …                 …            …               …          …
     1594            6.2             0.600         0.08             2.0      0.090
     1595            5.9             0.550         0.10             2.2      0.062
```

```
1596           6.3              0.510        0.13             2.3       0.076
1597           5.9              0.645        0.12             2.0       0.075
1598           6.0              0.310        0.47             3.6       0.067

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    11.0                  34.0  0.99780  3.51       0.56
1                    25.0                  67.0  0.99680  3.20       0.68
2                    15.0                  54.0  0.99700  3.26       0.65
3                    17.0                  60.0  0.99800  3.16       0.58
4                    11.0                  34.0  0.99780  3.51       0.56
…                    …                     …       …      …         …
1594                 32.0                  44.0  0.99490  3.45       0.58
1595                 39.0                  51.0  0.99512  3.52       0.76
1596                 29.0                  40.0  0.99574  3.42       0.75
1597                 32.0                  44.0  0.99547  3.57       0.71
1598                 18.0                  42.0  0.99549  3.39       0.66

      alcohol  quality
0         9.4        5
1         9.8        5
2         9.8        5
3         9.8        6
4         9.4        5
…         …      …
1594     10.5        5
1595     11.2        6
1596     11.0        6
1597     10.2        5
1598     11.0        6

[1599 rows x 12 columns]
```

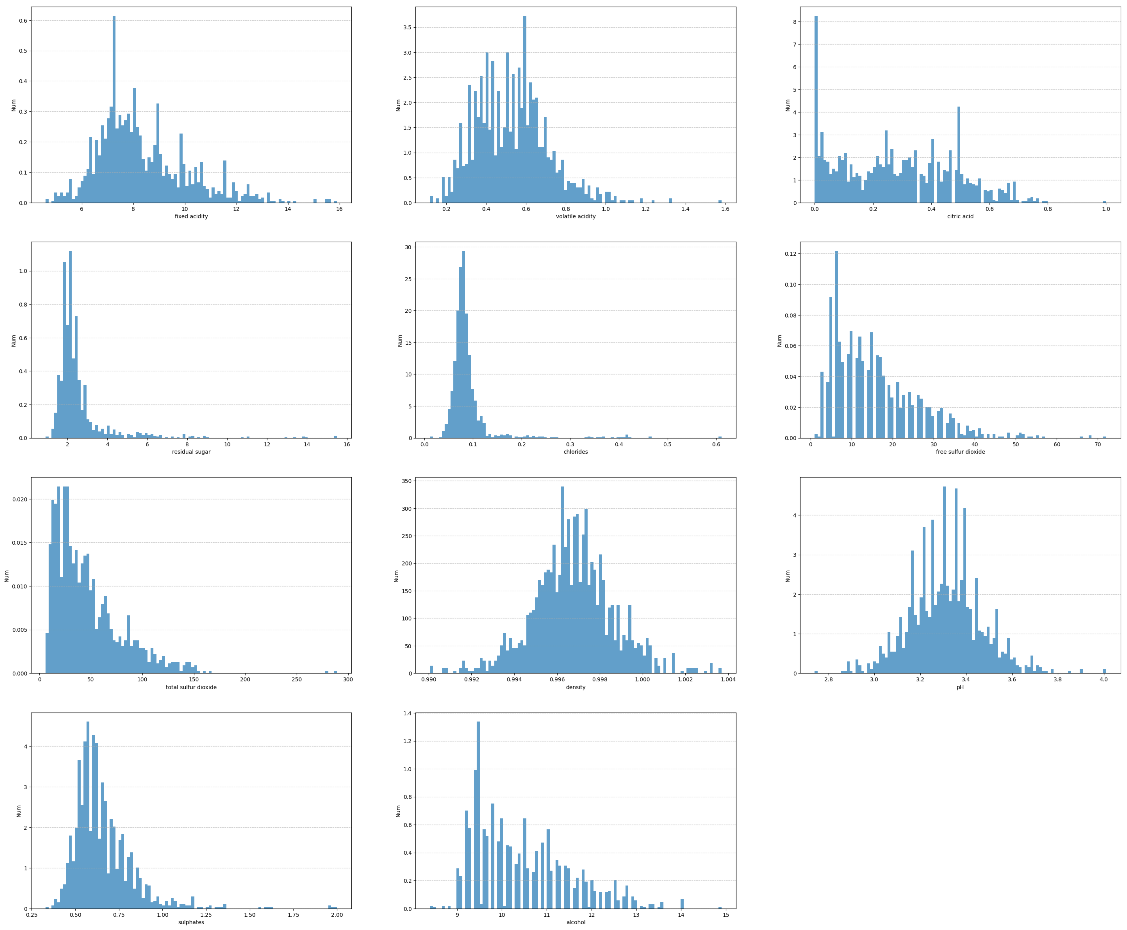
[5]:
```
missing_values = data.isnull().sum()
print("   \n", missing_values)
```

```
 fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
```

```
quality                 0
dtype: int64
```

[6]:
```
Continuous_value_column=[col for col in data.columns if col != 'quality']
```

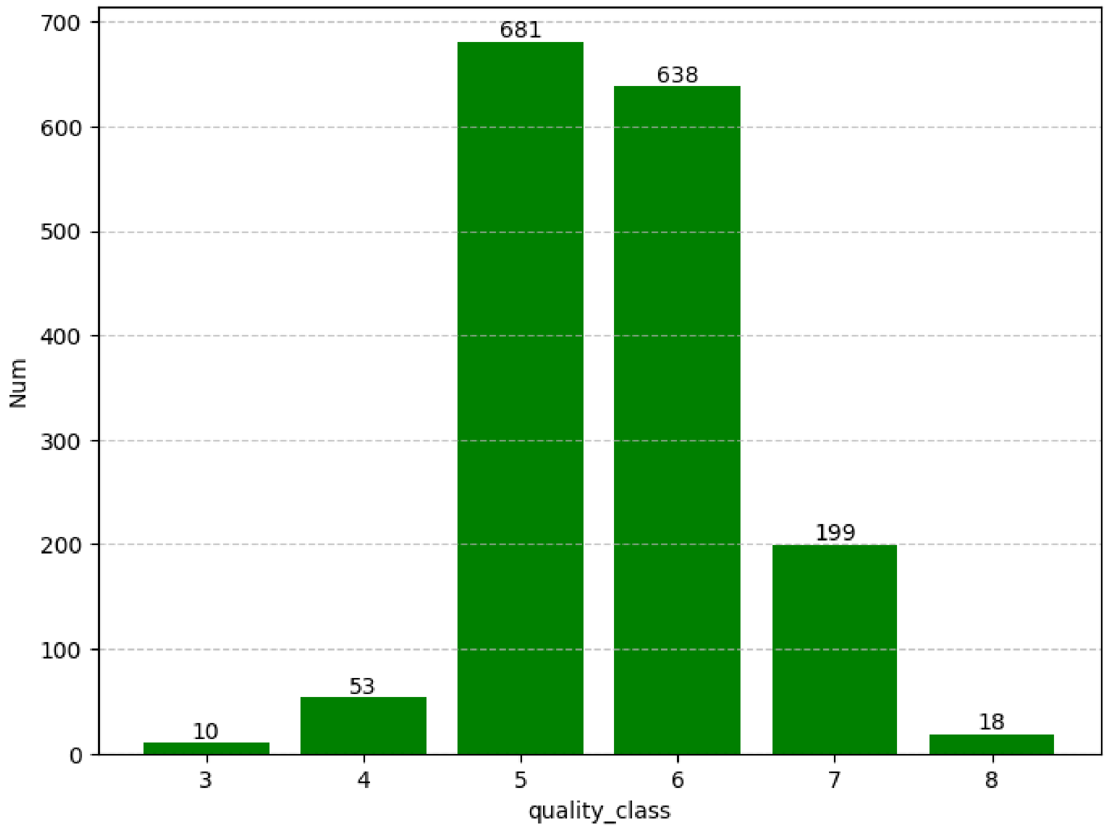
[7]:
```
plt.figure(figsize=(36,30))
for i,col in enumerate(Continuous_value_column):
    plt.subplot(4,3,i+1)
    plt.hist(data[col], bins=100, alpha=0.7, label=col, density=True)
    plt.xlabel(f'{col}')
    plt.ylabel('Num')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



[8]:
```
counter=Counter(data['quality'])
values=list(counter.values())
label = list(counter.keys())
plt.figure(figsize=(8,6))
bars=plt.bar(label, values, color='green',tick_label=label)
```

```
for bar in bars:
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height(),
        f'{bar.get_height()}',
        ha='center', va='bottom', fontsize=10
    )
plt.xlabel('quality_class')
plt.ylabel('Num')
plt.grid(axis='y', linestyle='--', alpha=0.7)

print(label)
plt.show()
```

```
[5, 6, 7, 4, 8, 3]
```



[11]:
```
threshold = 3

#     quality
#
```

```
#
#      quality

columns_to_clean = data.drop(columns=['quality'])

mean = columns_to_clean.mean()
std = columns_to_clean.std()

outliers = ((columns_to_clean - mean).abs() > threshold * std).any(axis=1)

cleaned_data = data[~outliers].copy()
cleaned_data = cleaned_data.reset_index(drop=True)
```

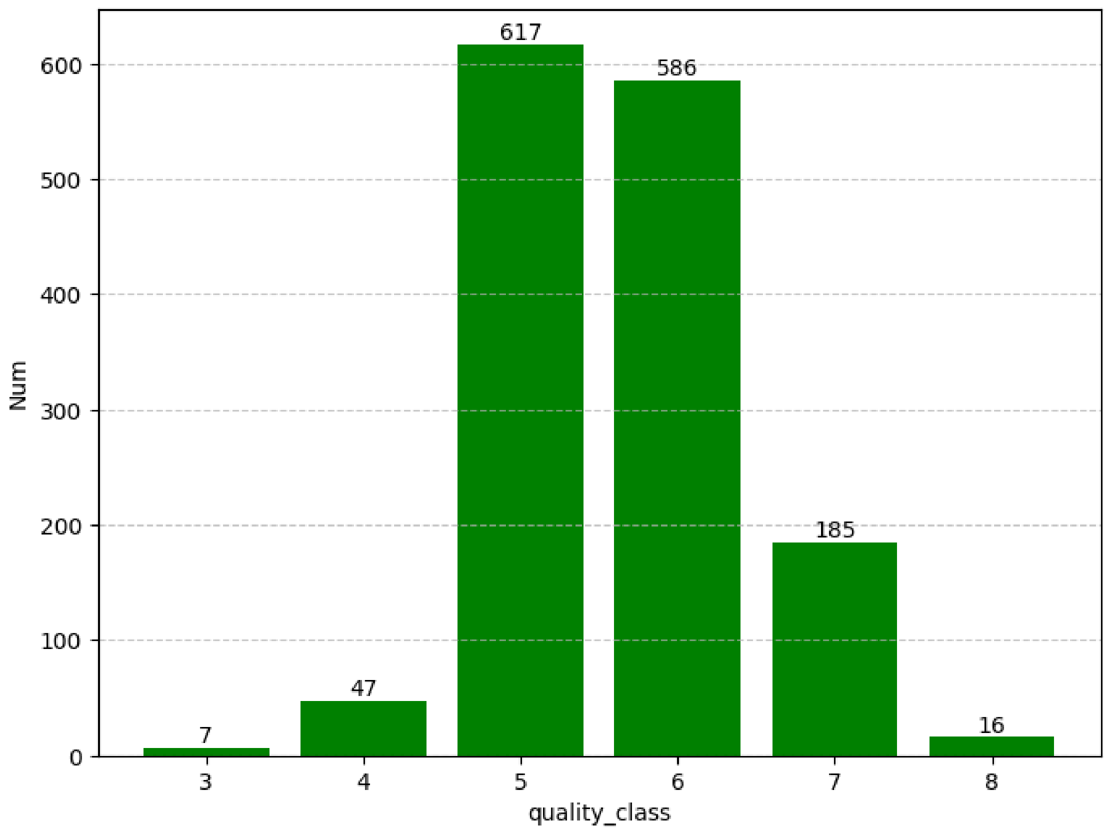
[12]:
```
cleaned_data
```

[12]:
```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0               7.4             0.700         0.00             1.9      0.076
1               7.8             0.880         0.00             2.6      0.098
2               7.8             0.760         0.04             2.3      0.092
3              11.2             0.280         0.56             1.9      0.075
4               7.4             0.700         0.00             1.9      0.076
…               …                …            …               …          …
1453            6.2             0.600         0.08             2.0      0.090
1454            5.9             0.550         0.10             2.2      0.062
1455            6.3             0.510         0.13             2.3      0.076
1456            5.9             0.645         0.12             2.0      0.075
1457            6.0             0.310         0.47             3.6      0.067

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    11.0                  34.0  0.99780  3.51       0.56
1                    25.0                  67.0  0.99680  3.20       0.68
2                    15.0                  54.0  0.99700  3.26       0.65
3                    17.0                  60.0  0.99800  3.16       0.58
4                    11.0                  34.0  0.99780  3.51       0.56
…                   …                    …      …     …        …
1453                 32.0                  44.0  0.99490  3.45       0.58
1454                 39.0                  51.0  0.99512  3.52       0.76
1455                 29.0                  40.0  0.99574  3.42       0.75
1456                 32.0                  44.0  0.99547  3.57       0.71
1457                 18.0                  42.0  0.99549  3.39       0.66

      alcohol  quality
0         9.4        5
1         9.8        5
2         9.8        5
3         9.8        6
4         9.4        5
```

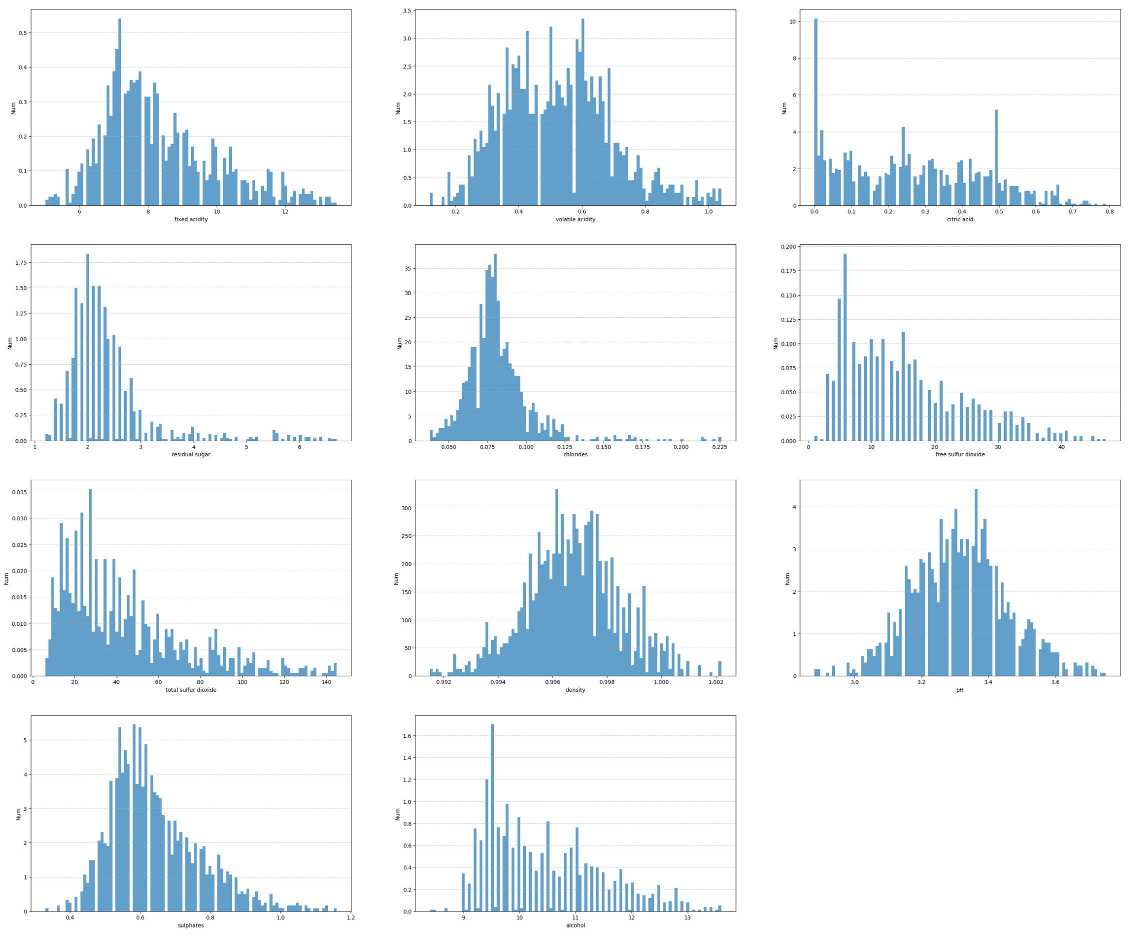
```
…        …        …
1453      10.5        5
1454      11.2        6
1455      11.0        6
1456      10.2        5
1457      11.0        6

[1458 rows x 12 columns]
```

[13]:
```python
counter=Counter(cleaned_data['quality'])
values=list(counter.values())
label = list(counter.keys())
plt.figure(figsize=(8,6))
bars=plt.bar(label, values, color='green',tick_label=label)

for bar in bars:
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height(),
        f'{bar.get_height()}',
        ha='center', va='bottom', fontsize=10
    )
plt.xlabel('quality_class')
plt.ylabel('Num')
plt.grid(axis='y', linestyle='--', alpha=0.7)

print(label)
plt.show()
```

```
[5, 6, 7, 4, 8, 3]
```



[14]:
```
plt.figure(figsize=(36,30))
for i,col in enumerate(Continuous_value_column):
    plt.subplot(4,3,i+1)
    plt.hist(cleaned_data[col], bins=100, alpha=0.7, label=col, density=True)
    plt.xlabel(f'{col}')
    plt.ylabel('Num')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



[21]:
```
train_val_data, test_data = train_test_split(cleaned_data, test_size=0.2,␣
 ↪random_state=55,stratify=cleaned_data['quality'])
```

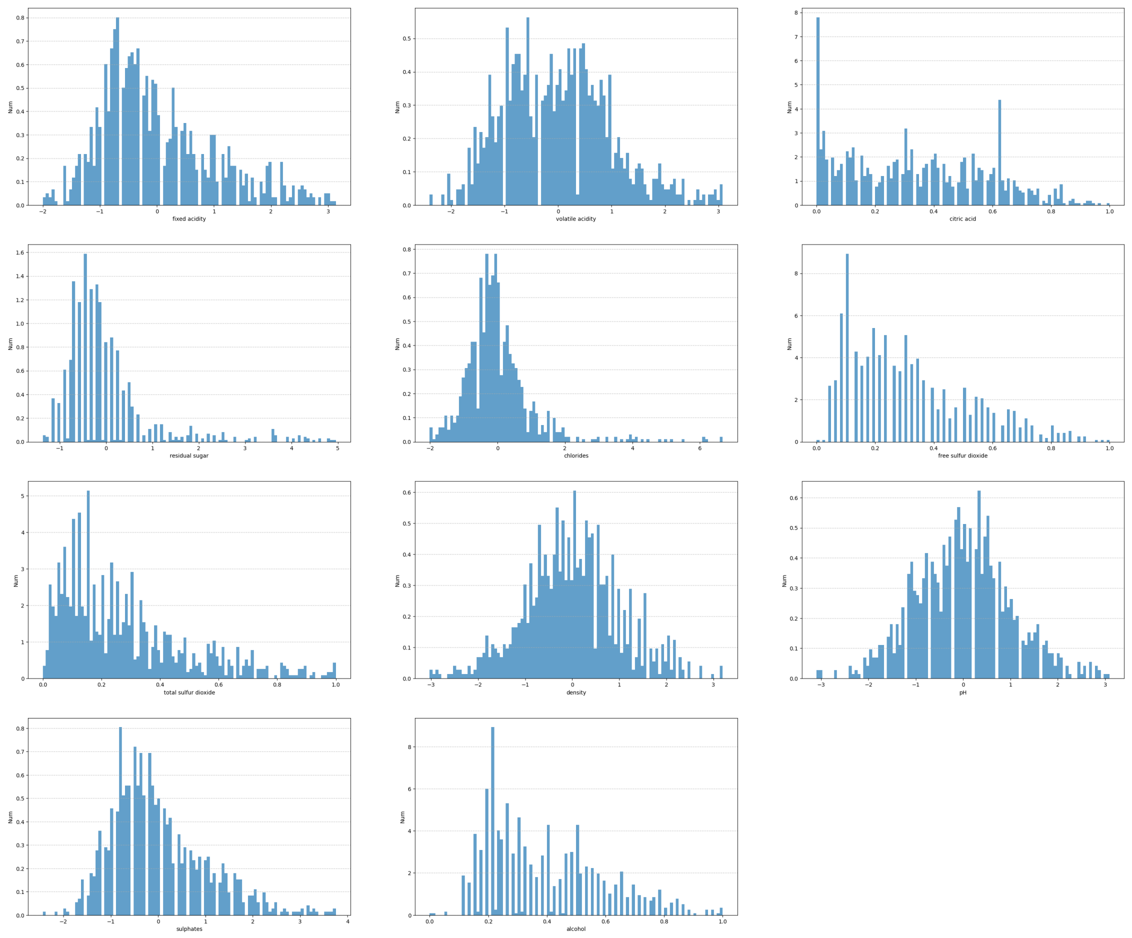
[22]:
```
#
'''    citric acid free sulfur dioxide total sulfur dioxide alcohol'''

Columns_Minmax=['citric acid','free sulfur dioxide','total sulfur␣
 ↪dioxide','alcohol']
Columns_normalize=[col for col in Continuous_value_column if col not in␣
 ↪Columns_Minmax]
```

[23]:
```
scaler = StandardScaler()
train_val_data[Columns_normalize]= scaler.
 ↪fit_transform(train_val_data[Columns_normalize])
test_data[Columns_normalize]= scaler.fit_transform(test_data[Columns_normalize])
```

[24]:
```
scaler_2 = MinMaxScaler()
```

```
train_val_data[Columns_Minmax]= scaler_2.
 ↪fit_transform(train_val_data[Columns_Minmax])
test_data[Columns_Minmax]= scaler_2.fit_transform(test_data[Columns_Minmax])
```

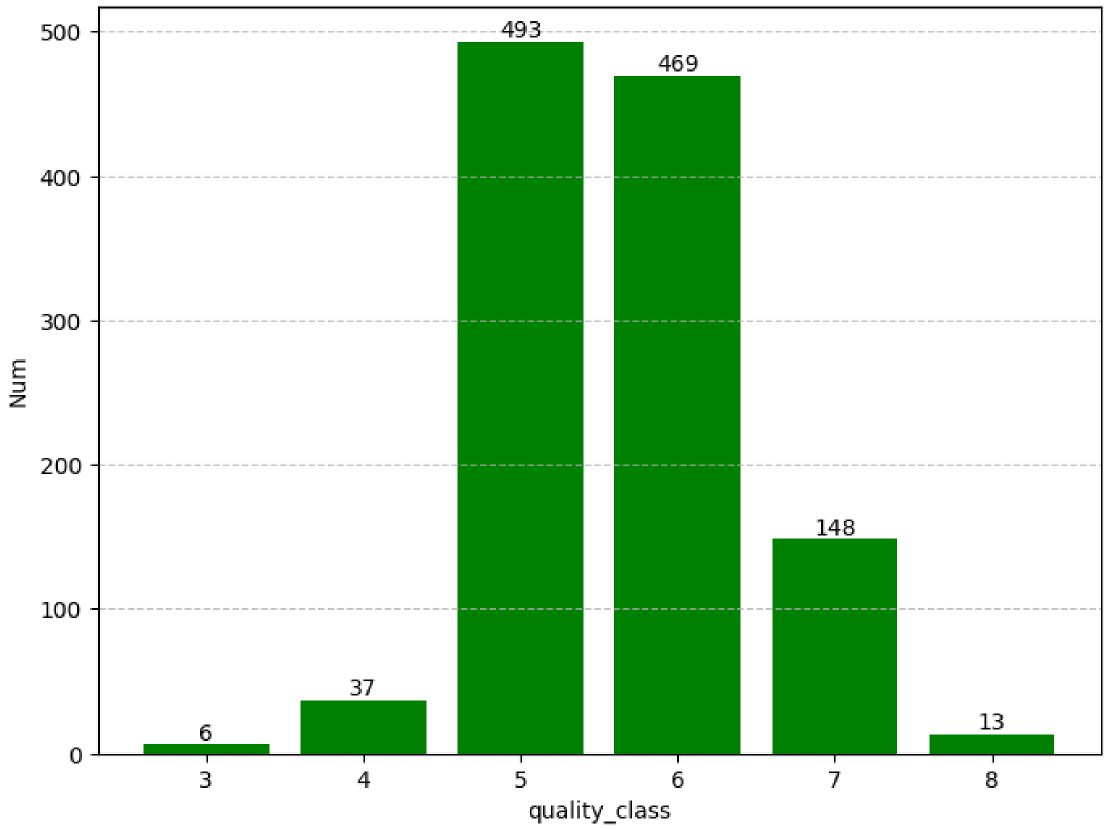
```
[25]: plt.figure(figsize=(36,30))
for i,col in enumerate(Continuous_value_column):
    plt.subplot(4,3,i+1)
    plt.hist(train_val_data[col], bins=100, alpha=0.7, label=col, density=True)
    plt.xlabel(f'{col}')
    plt.ylabel('Num')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



```
[26]: counter=Counter(train_val_data['quality'])
values=list(counter.values())
label = list(counter.keys())
plt.figure(figsize=(8,6))
bars=plt.bar(label, values, color='green',tick_label=label)
```

```
#
for bar in bars:
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height(),
        f'{bar.get_height()}',
        ha='center', va='bottom', fontsize=10
    )
plt.xlabel('quality_class')
plt.ylabel('Num')
plt.grid(axis='y', linestyle='--', alpha=0.7)

print(label)
plt.show()
```

```
[5, 6, 7, 4, 8, 3]
```



[27]:
```
train_val_data.to_csv("./dataset/train_val_data.csv", index=False)
test_data.to_csv("./dataset/test_data.csv", index=False)
```

[28]:
```
correlation_matrix = train_val_data.corr()
```

[29]:
```
correlation_matrix
```

[29]:
```
                      fixed acidity  volatile acidity  citric acid  \
fixed acidity              1.000000         -0.298442     0.694411
volatile acidity          -0.298442          1.000000    -0.589030
citric acid                0.694411         -0.589030     1.000000
residual sugar             0.176996          0.042129     0.174421
chlorides                  0.193477          0.074038     0.099016
free sulfur dioxide       -0.154162          0.009925    -0.070989
total sulfur dioxide      -0.095100          0.103610     0.021995
density                    0.667733         -0.002568     0.377311
pH                        -0.698222          0.259103    -0.506599
sulphates                  0.200222         -0.358433     0.299544
alcohol                   -0.027732         -0.210403     0.158811
quality                    0.130129         -0.367388     0.228575

                      residual sugar  chlorides  free sulfur dioxide  \
fixed acidity               0.176996   0.193477            -0.154162
volatile acidity            0.042129   0.074038             0.009925
citric acid                 0.174421   0.099016            -0.070989
residual sugar              1.000000   0.125540            -0.005724
chlorides                   0.125540   1.000000            -0.071603
free sulfur dioxide        -0.005724  -0.071603             1.000000
total sulfur dioxide        0.104507   0.037607             0.654567
density                     0.351051   0.340064            -0.050243
pH                         -0.073379  -0.177122             0.117185
sulphates                   0.064467   0.019450             0.072481
alcohol                     0.160311  -0.208257            -0.058125
quality                     0.074180  -0.128932            -0.078138

                      total sulfur dioxide   density        pH  sulphates  \
fixed acidity                    -0.095100  0.667733 -0.698222   0.200222
volatile acidity                  0.103610 -0.002568  0.259103  -0.358433
citric acid                       0.021995  0.377311 -0.506599   0.299544
residual sugar                    0.104507  0.351051 -0.073379   0.064467
chlorides                         0.037607  0.340064 -0.177122   0.019450
free sulfur dioxide               0.654567 -0.050243  0.117185   0.072481
total sulfur dioxide              1.000000  0.123091 -0.010719  -0.038337
density                           0.123091  1.000000 -0.316929   0.116521
pH                               -0.010719 -0.316929  1.000000  -0.042781
sulphates                        -0.038337  0.116521 -0.042781   1.000000
alcohol                          -0.253665 -0.467740  0.156574   0.236423
quality                          -0.242274 -0.185747 -0.081741   0.377663

                       alcohol   quality
fixed acidity        -0.027732  0.130129
volatile acidity     -0.210403 -0.367388
```

```
citric acid           0.158811  0.228575
residual sugar        0.160311  0.074180
chlorides            -0.208257 -0.128932
free sulfur dioxide  -0.058125 -0.078138
total sulfur dioxide -0.253665 -0.242274
density              -0.467740 -0.185747
pH                    0.156574 -0.081741
sulphates             0.236423  0.377663
alcohol               1.000000  0.497556
quality               0.497556  1.000000
```

[30]:
```
#       0.6
threshold = 0.6
high_corr = correlation_matrix.abs() > threshold
high_corr_pairs = high_corr.where(np.triu(np.ones(high_corr.shape), k=1).
 ↪astype(bool))

feature_pairs = [
    (correlation_matrix.index[i], correlation_matrix.columns[j],␣
 ↪correlation_matrix.iloc[i, j])
    for i in range(len(high_corr_pairs))
    for j in range(i + 1, len(high_corr_pairs))
    if high_corr_pairs.iloc[i, j]
]

print("     0.6    ")
for pair in feature_pairs:
    print(f"{pair[0]}   {pair[1]}     {pair[2]:.2f}")
```

```
     0.6
fixed acidity   citric acid     0.69
fixed acidity   density     0.67
fixed acidity   pH     -0.70
free sulfur dioxide   total sulfur dioxide     0.65
```

[57]:
```
#
train_val_data_pca=pd.DataFrame()
test_data_pca=pd.DataFrame()
pca = PCA(n_components=1)
for pair in feature_pairs:
    train_val_data_pca_0=train_val_data[[pair[0],pair[1]]].copy()
    test_data_pca_0=test_data[[pair[0],pair[1]]].copy()
    train_val_data_pca[pair[0]+pair[1]]=pd.Series(pca.
 ↪fit_transform(train_val_data_pca_0).ravel(),index=train_val_data_pca_0.index)
    test_data_pca[pair[0]+pair[1]]=pd.Series(pca.fit_transform(test_data_pca_0).
 ↪ravel(),index=test_data_pca_0.index)
```

```
[59]: PCA_feature=['fixed acidity','citric acid','density','pH','free sulfur␣
       ↪dioxide','total sulfur dioxide']
```

```
[60]: Columns_save=[col for col in data.columns if col not in PCA_feature]
```

```
[61]: train_val_data_pca[Columns_save]=train_val_data[Columns_save]
      test_data_pca[Columns_save]=test_data[Columns_save]
```

```
[67]: train_val_data_pca.to_csv("./dataset/train_val_data_pca-9.csv", index=False)
      test_data_pca.to_csv("./dataset/test_data_pca-9.csv", index=False)
```

```
[71]: #    'fixed acidity','citric acid','density','pH'   'free sulfur␣
       ↪dioxide','total sulfur dioxide'
      train_val_data_pca_7=pd.DataFrame()
      test_data_pca_7=pd.DataFrame()
      pca= PCA(n_components=1)
      train_val_data_pca_1=train_val_data[['fixed acidity','citric␣
       ↪acid','density','pH']].copy()
      test_data_pca_1=test_data[['fixed acidity','citric acid','density','pH']].copy()
      train_val_data_pca_7['fixed acidity citric acid density pH']=pd.Series(pca.
       ↪fit_transform(train_val_data_pca_1).ravel(),index=train_val_data_pca_1.index)
      test_data_pca_7['fixed acidity citric acid density pH']=pd.Series(pca.
       ↪fit_transform(test_data_pca_1).ravel(),index=test_data_pca_1.index)
      train_val_data_pca_2=train_val_data[['free sulfur dioxide','total sulfur␣
       ↪dioxide']].copy()
      test_data_pca_2=test_data[['free sulfur dioxide','total sulfur dioxide']].copy()
      train_val_data_pca_7['sulfur dioxide']=pd.Series(pca.
       ↪fit_transform(train_val_data_pca_2).ravel(),index=train_val_data_pca_2.index)
      test_data_pca_7['sulfur dioxide']=pd.Series(pca.fit_transform(test_data_pca_2).
       ↪ravel(),index=test_data_pca_2.index)
      train_val_data_pca_7[Columns_save]=train_val_data[Columns_save]
      test_data_pca_7[Columns_save]=test_data[Columns_save]
```

```
[72]: train_val_data_pca_7
```

```
[72]:       fixed acidity citric acid density pH  sulfur dioxide  volatile acidity  \
      1229                              -1.670005       -0.217017         -1.208490
      591                                3.624173        0.047082         -0.732242
      338                               -0.137035        0.134381          0.101193
      859                                0.557986       -0.337845         -1.089428
      643                                1.516722        0.617306         -0.255994
      …                                         …               …                 …
      561                                0.031408        0.278945          0.964393
      517                                1.874040        0.018092         -1.268021
      1280                              -1.795679       -0.240754         -0.315525
      349                                0.852800       -0.105074          1.440641
```

```
1003                            -0.519800       -0.236408        -1.744270

      residual sugar  chlorides  sulphates   alcohol  quality
1229       -0.674715   0.207058  -1.582975  0.326923        5
591        -0.444764   0.484246   0.118675  0.346154        5
338        -0.444764   0.299454   0.737457  0.153846        6
859        -0.329789  -1.363670   0.350718  0.711538        7
643         0.532524   0.576642  -0.886846  0.153846        5
…                …         …         …         …       …
561        -0.099839   1.362006  -0.268064  0.173077        5
517        -0.214814  -0.855493  -1.041541  0.269231        6
1280        0.015136  -0.485910   0.660109  0.423077        6
349        -0.099839  -0.809295  -0.732150  0.192308        5
1003       -0.559739  -0.578306  -0.809498  0.750000        6

[1166 rows x 8 columns]
```

[73]:
```
train_val_data_pca_7.to_csv("./dataset/train_val_data_pca-7.csv", index=False)
test_data_pca_7.to_csv("./dataset/test_data_pca-7.csv", index=False)
```

[74]:
```
#
pca_5=PCA(n_components=5)
train_val_data_pca_5=pd.DataFrame()
test_data_pca_5=pd.DataFrame()
train_val_data_pca_3=train_val_data[Continuous_value_column].copy()
test_data_pca_3=test_data[Continuous_value_column].copy()
```

[85]:
```
train_val_data_pca_5[['feature 1','feature 2','feature 3','feature 4','feature␣
 ↪5']]=  pd.DataFrame(pca_5.
 ↪fit_transform(train_val_data_pca_3),train_val_data_pca_3.index)
test_data_pca_5[['feature 1','feature 2','feature 3','feature 4','feature␣
 ↪5']]=pd.DataFrame(pca_5.fit_transform(test_data_pca_3),test_data_pca_3.index)
```

[86]:
```
train_val_data_pca_5['quality']=train_val_data['quality']
test_data_pca_5['quality']=test_data['quality']
```

[90]:
```
train_val_data_pca_5.to_csv("./dataset/train_val_data_pca-5.csv", index=False)
test_data_pca_5.to_csv("./dataset/test_data_pca-5.csv", index=False)
```

[ ]:
```

```

December 16, 2024

[112]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, KFold
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import seaborn as sns
from matplotlib import cm
from matplotlib.colors import to_hex
```

[ ]:
```

```

[107]:
```
file_path="./dataset_encoder_6/train_val_data_pca-9.csv"
file_path_2="./dataset_encoder_6/test_data_pca-9.csv"
data_train_valid = pd.read_csv(file_path)
data_test=pd.read_csv(file_path_2)
```

[108]:
```
features_columns=[col for col in data_train_valid.columns if col != 'quality']
```

[109]:
```
x_train_val=data_train_valid[features_columns]
y_train_val=data_train_valid['quality']
x_test=data_test[features_columns]
y_test=data_test['quality']
```

[110]:
```
hidden_nodes_sizes=np.arange(10,110,5)

kf = KFold(n_splits=5, shuffle=True, random_state=55)
accuracies_result={}
accuracies_individual={}
for hidden_node_size in hidden_nodes_sizes:
    ANN_model = MLPClassifier(solver='lbfgs',␣
 ↪alpha=1e-5,hidden_layer_sizes=(hidden_node_size, 2), random_state=55)
    accuracy=0
    accuracy_individual=[]
    for train_index, val_index in kf.split(x_train_val):
        x_train, x_val = x_train_val.iloc[train_index], x_train_val.
 ↪iloc[val_index]
```

```
        y_train, y_val = y_train_val.iloc[train_index], y_train_val.
↪iloc[val_index]
        ANN_model.fit(x_train,y_train)
        y_pred=ANN_model.predict(x_val)
        accuracy+=accuracy_score(y_val,y_pred)
        accuracy_individual.append(accuracy_score(y_val,y_pred))
    ␣
↪accuracies_individual[f'hidden_nodes_{hidden_node_size}']=accuracy_individual
    accuracies_result[f'hidden_nodes_{hidden_node_size}']=accuracy/5
print(accuracies_individual)
print(accuracies_result)
```

```
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

```
{'hidden_nodes_10': [0.8846153846153846, 0.8283261802575107, 0.8841201716738197,
0.8540772532188842, 0.8755364806866953], 'hidden_nodes_15': [0.8632478632478633,
0.8369098712446352, 0.9012875536480687, 0.8369098712446352, 0.8841201716738197],
'hidden_nodes_20': [0.8931623931623932, 0.8454935622317596, 0.8497854077253219,
0.8583690987124464, 0.9055793991416309], 'hidden_nodes_25': [0.8931623931623932,
0.8154506437768241, 0.8412017167381974, 0.8326180257510729, 0.8669527896995708],
'hidden_nodes_30': [0.8632478632478633, 0.8669527896995708, 0.9141630901287554,
0.871244635193133, 0.8626609442060086], 'hidden_nodes_35': [0.8888888888888888,
0.8369098712446352, 0.8841201716738197, 0.8454935622317596, 0.9098712446351931],
'hidden_nodes_40': [0.9017094017094017, 0.871244635193133, 0.9012875536480687,
0.8626609442060086, 0.8841201716738197], 'hidden_nodes_45': [0.8803418803418803,
0.8798283261802575, 0.871244635193133, 0.8798283261802575, 0.9098712446351931],
'hidden_nodes_50': [0.8803418803418803, 0.8626609442060086, 0.9055793991416309,
0.8626609442060086, 0.8841201716738197], 'hidden_nodes_55': [0.8803418803418803,
0.8884120171673819, 0.8884120171673819, 0.8583690987124464, 0.8626609442060086],
'hidden_nodes_60': [0.8931623931623932, 0.871244635193133, 0.9012875536480687,
0.8755364806866953, 0.871244635193133], 'hidden_nodes_65': [0.9017094017094017,
0.8583690987124464, 0.9184549356223176, 0.8755364806866953, 0.8927038626609443],
'hidden_nodes_70': [0.8931623931623932, 0.8626609442060086, 0.9098712446351931,
0.8497854077253219, 0.8669527896995708], 'hidden_nodes_75': [0.8675213675213675,
0.8669527896995708, 0.8669527896995708, 0.871244635193133, 0.8927038626609443],
'hidden_nodes_80': [0.8974358974358975, 0.8497854077253219, 0.9227467811158798,
0.8583690987124464, 0.8798283261802575], 'hidden_nodes_85': [0.8846153846153846,
0.8583690987124464, 0.8969957081545065, 0.8798283261802575, 0.8927038626609443],
'hidden_nodes_90': [0.8717948717948718, 0.8626609442060086, 0.9098712446351931,
0.8626609442060086, 0.871244635193133], 'hidden_nodes_95': [0.8803418803418803,
0.8540772532188842, 0.8755364806866953, 0.8798283261802575, 0.8927038626609443],
'hidden_nodes_100': [0.9145299145299145, 0.8755364806866953, 0.9055793991416309,
0.8497854077253219, 0.9098712446351931], 'hidden_nodes_105':
[0.9145299145299145, 0.8412017167381974, 0.8969957081545065, 0.8669527896995708,
0.8841201716738197]}
{'hidden_nodes_10': 0.865335094090459, 'hidden_nodes_15': 0.8644950662118044,
'hidden_nodes_20': 0.8704779721947103, 'hidden_nodes_25': 0.8498771138256117,
'hidden_nodes_30': 0.8756538644950662, 'hidden_nodes_35': 0.8730567477348593,
'hidden_nodes_40': 0.8842045412860863, 'hidden_nodes_45': 0.8842228825061442,
'hidden_nodes_50': 0.8790726679138695, 'hidden_nodes_55': 0.8756391915190198,
'hidden_nodes_60': 0.8824951395766846, 'hidden_nodes_65': 0.889354755878361,
'hidden_nodes_70': 0.8764865558856976, 'hidden_nodes_75': 0.8730750889549173,
'hidden_nodes_80': 0.8816331022339605, 'hidden_nodes_85': 0.8825024760647079,
'hidden_nodes_90': 0.8756465280070429, 'hidden_nodes_95': 0.8764975606177323,
'hidden_nodes_100': 0.8910604893437511, 'hidden_nodes_105': 0.8807600601592018}
```

```
e:\python\python\envs\pytorch\lib\site-
```

```
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

[111]:
```python
print(accuracies_individual)
print(accuracies_result)
```

```
{'hidden_nodes_10': [0.8846153846153846, 0.8283261802575107, 0.8841201716738197,
0.8540772532188842, 0.8755364806866953], 'hidden_nodes_15': [0.8632478632478633,
0.8369098712446352, 0.9012875536480687, 0.8369098712446352, 0.8841201716738197],
'hidden_nodes_20': [0.8931623931623932, 0.8454935622317596, 0.8497854077253219,
0.8583690987124464, 0.9055793991416309], 'hidden_nodes_25': [0.8931623931623932,
0.8154506437768241, 0.8412017167381974, 0.8326180257510729, 0.8669527896995708],
'hidden_nodes_30': [0.8632478632478633, 0.8669527896995708, 0.9141630901287554,
0.871244635193133, 0.8626609442060086], 'hidden_nodes_35': [0.8888888888888888,
0.8369098712446352, 0.8841201716738197, 0.8454935622317596, 0.9098712446351931],
'hidden_nodes_40': [0.9017094017094017, 0.871244635193133, 0.9012875536480687,
0.8626609442060086, 0.8841201716738197], 'hidden_nodes_45': [0.8803418803418803,
0.8798283261802575, 0.871244635193133, 0.8798283261802575, 0.9098712446351931],
'hidden_nodes_50': [0.8803418803418803, 0.8626609442060086, 0.9055793991416309,
0.8626609442060086, 0.8841201716738197], 'hidden_nodes_55': [0.8803418803418803,
0.8884120171673819, 0.8884120171673819, 0.8583690987124464, 0.8626609442060086],
'hidden_nodes_60': [0.8931623931623932, 0.871244635193133, 0.9012875536480687,
0.8755364806866953, 0.871244635193133], 'hidden_nodes_65': [0.9017094017094017,
0.8583690987124464, 0.9184549356223176, 0.8755364806866953, 0.8927038626609443],
'hidden_nodes_70': [0.8931623931623932, 0.8626609442060086, 0.9098712446351931,
0.8497854077253219, 0.8669527896995708], 'hidden_nodes_75': [0.8675213675213675,
0.8669527896995708, 0.8669527896995708, 0.871244635193133, 0.8927038626609443],
'hidden_nodes_80': [0.8974358974358975, 0.8497854077253219, 0.9227467811158798,
0.8583690987124464, 0.8798283261802575], 'hidden_nodes_85': [0.8846153846153846,
0.8583690987124464, 0.8969957081545065, 0.8798283261802575, 0.8927038626609443],
'hidden_nodes_90': [0.8717948717948718, 0.8626609442060086, 0.9098712446351931,
0.8626609442060086, 0.871244635193133], 'hidden_nodes_95': [0.8803418803418803,
0.8540772532188842, 0.8755364806866953, 0.8798283261802575, 0.8927038626609443],
'hidden_nodes_100': [0.9145299145299145, 0.8755364806866953, 0.9055793991416309,
0.8497854077253219, 0.9098712446351931], 'hidden_nodes_105':
[0.9145299145299145, 0.8412017167381974, 0.8969957081545065, 0.8669527896995708,
0.8841201716738197]}
{'hidden_nodes_10': 0.865335094090459, 'hidden_nodes_15': 0.8644950662118044,
'hidden_nodes_20': 0.8704779721947103, 'hidden_nodes_25': 0.8498771138256117,
'hidden_nodes_30': 0.8756538644950662, 'hidden_nodes_35': 0.8730567477348593,
'hidden_nodes_40': 0.8842045412860863, 'hidden_nodes_45': 0.8842228825061442,
'hidden_nodes_50': 0.8790726679138695, 'hidden_nodes_55': 0.8756391915190198,
'hidden_nodes_60': 0.8824951395766846, 'hidden_nodes_65': 0.889354755878361,
'hidden_nodes_70': 0.8764865558856976, 'hidden_nodes_75': 0.8730750889549173,
'hidden_nodes_80': 0.8816331022339605, 'hidden_nodes_85': 0.8825024760647079,
'hidden_nodes_90': 0.8756465280070429, 'hidden_nodes_95': 0.8764975606177323,
'hidden_nodes_100': 0.8910604893437511, 'hidden_nodes_105': 0.8807600601592018}
```

[ ]:

```
[106]: optimize_ANN_model= MLPClassifier(solver='lbfgs',␣
        ↪alpha=1e-5,hidden_layer_sizes=(100, 2), random_state=55)
       optimize_ANN_model.fit(x_train_val,y_train_val)
       y_pred=optimize_ANN_model.predict(x_test)
       accuracy_result=accuracy_score(y_test,y_pred)
       print(f"Test Accuracies:{accuracy_result}")
```

```
Test Accuracies:0.2773972602739726

e:\python\python\envs\pytorch\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:546:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

December 16, 2024

[1]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
```

[3]:
```
train_datasets = [
    'train_val_data_pca-9.csv',
    'train_val_data_pca-7.csv',
    'train_val_data_pca-5.csv',
    'train_val_data.csv'
]

test_datasets = [
    'test_data_pca-9.csv',
    'test_data_pca-7.csv',
    'test_data_pca-5.csv',
    'test_data.csv'
]

param_grid = {
    'n_estimators': [150, 175, 200, 225, 250],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

results = {}
```

[5]:
```
#

train_data_pca9 = pd.read_csv(f'dataset6/{train_datasets[0]}')
X_train_pca9 = train_data_pca9.drop('quality', axis=1)
y_train_pca9 = train_data_pca9['quality']
```

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

rf_grid = GridSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_grid=param_grid,
    cv=cv,
    scoring='accuracy',
    n_jobs=-1
)

rf_grid.fit(X_train_pca9, y_train_pca9)
```

```
[5]: GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=42, shuffle=True),
                  estimator=RandomForestClassifier(random_state=42), n_jobs=-1,
                  param_grid={'max_depth': [None, 10, 20, 30],
                              'min_samples_leaf': [1, 2, 4],
                              'min_samples_split': [2, 5, 10],
                              'n_estimators': [150, 175, 200, 225, 250]},
                  scoring='accuracy')
```

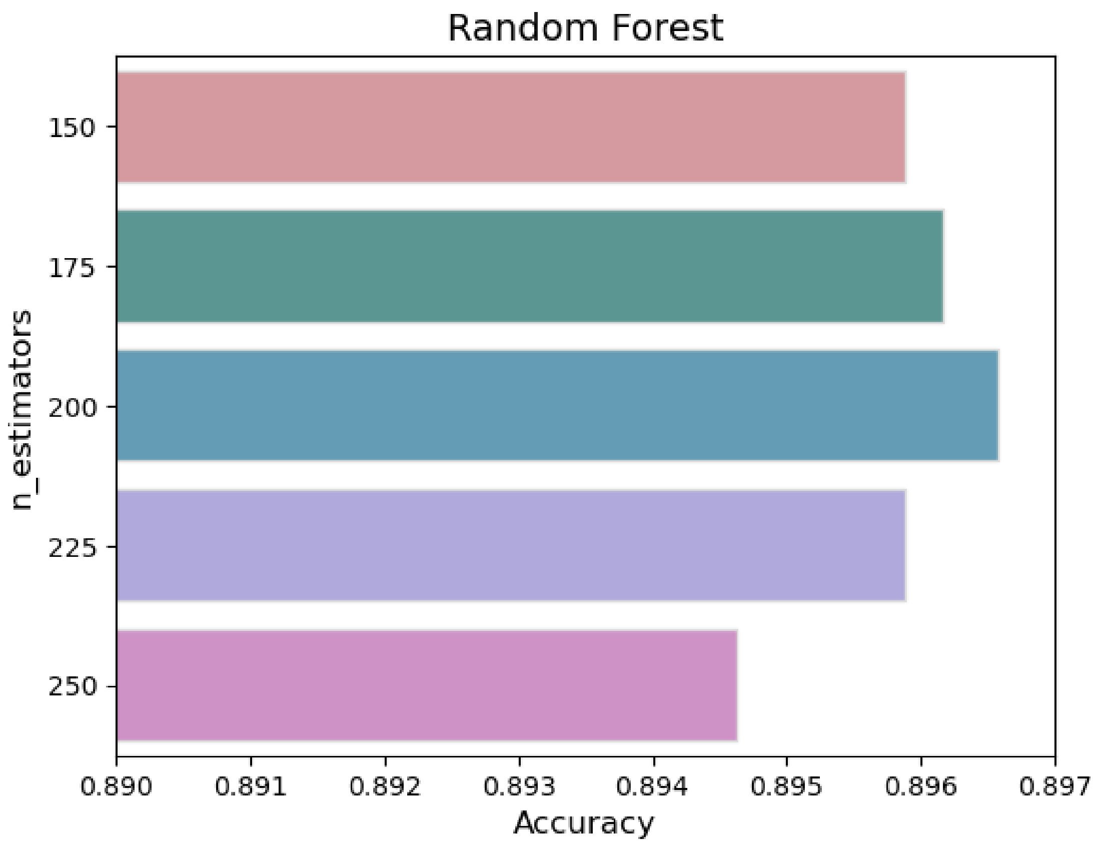
```
[35]: cv_results = rf_grid.cv_results_
n_estimators_values = [params['n_estimators'] for params in␣
 ↪cv_results['params']]
unique_n_estimators = sorted(set(n_estimators_values))

mean_scores = []
for n_est in unique_n_estimators:
    mask = [params['n_estimators'] == n_est for params in cv_results['params']]
    mean_acc = np.mean(cv_results['mean_test_score'][mask])
    mean_scores.append(mean_acc)

print("       ")
for n_est, acc in zip(unique_n_estimators, mean_scores):
    print(f"n_estimators={n_est}:    ={acc:.4f}")

#         n_estimators
best_n_est_index = np.argmax(mean_scores)
best_n_est_from_bar = unique_n_estimators[best_n_est_index]
print(f"\n        n_estimators={best_n_est_from_bar},␣
 ↪  ={mean_scores[best_n_est_index]:.4f}")
```

```
n_estimators=150:    =0.8959
n_estimators=175:    =0.8962
n_estimators=200:    =0.8966
n_estimators=225:    =0.8959
n_estimators=250:    =0.8946
```

```
        n_estimators=200,   =0.8966
```

[29]:
```
#  Seaborn barplot
import seaborn as sns

n_estimators_list = [str(x) for x in unique_n_estimators]
accuracy_values = mean_scores

hyper_parameter_optimze = {
    'n_estimators': n_estimators_list,
    'Accuracy': accuracy_values
}
hyper_parameter_optimze_frame = pd.DataFrame(hyper_parameter_optimze)

colorlist = ['#de9099', '#539f9a', '#57a1c3', '#aaa1e3', '#d988cd']

sns.barplot(x='Accuracy', y='n_estimators', data=hyper_parameter_optimze_frame,␣
  ↪palette=colorlist, edgecolor='#DDDDDD')

plt.xlim(0.8900, 0.8970)
plt.title("Random Forest", fontsize=14)
plt.xlabel("Accuracy", fontsize=12)
plt.ylabel("n_estimators", fontsize=12)

plt.show()
```

```
C:\Users\10940\AppData\Local\Temp\ipykernel_11468\1126308933.py:19:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='Accuracy', y='n_estimators',
data=hyper_parameter_optimze_frame, palette=colorlist, edgecolor='#DDDDDD')
```

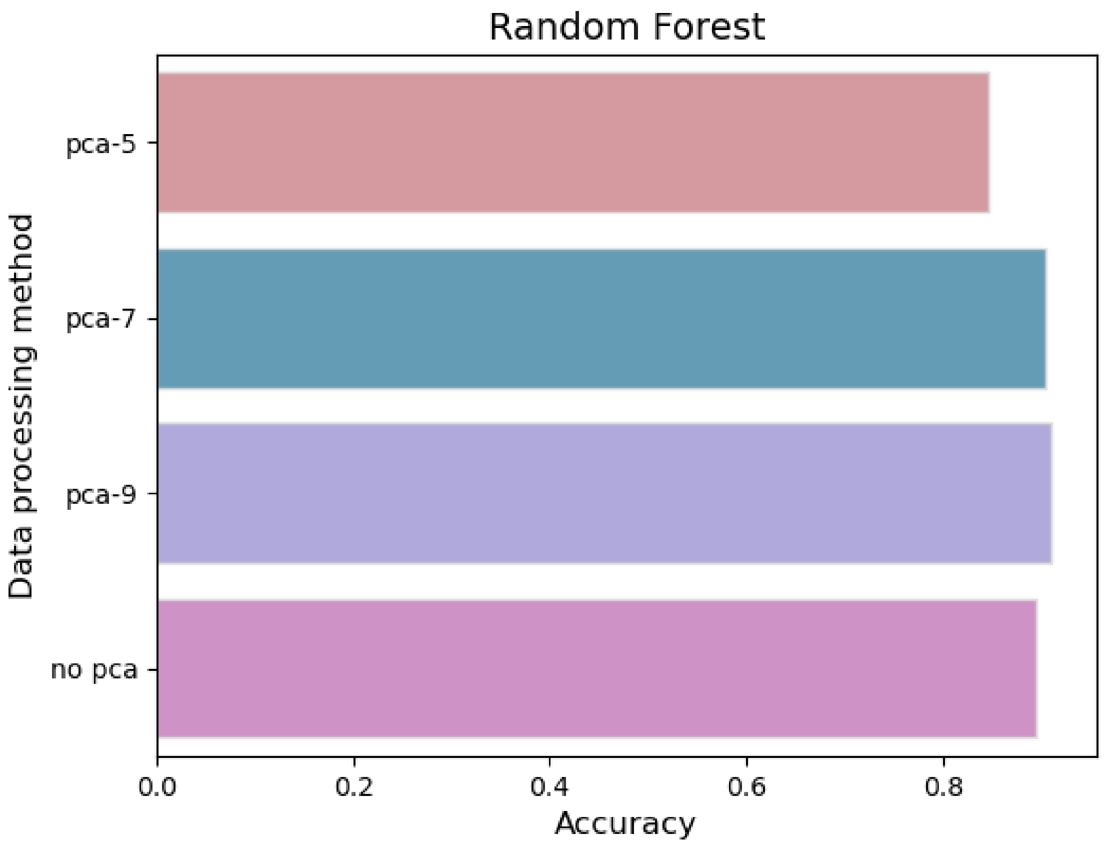


[23]:
```
best_params = rf_grid.best_params_.copy()
best_params['n_estimators'] = 200
custom_rf = RandomForestClassifier(**best_params, random_state=42)

#  pca-9 pca-7 pca-5  no  pca(ram)

for train_name, test_name in zip(train_datasets, test_datasets):

    train_data = pd.read_csv(f'dataset6/{train_name}')
    test_data = pd.read_csv(f'dataset6/{test_name}')
    X_train = train_data.drop('quality', axis=1)
    y_train = train_data['quality']
    X_test = test_data.drop('quality', axis=1)
    y_test = test_data['quality']

    custom_rf.fit(X_train, y_train)
    y_pred = custom_rf.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    clf_report = classification_report(y_test, y_pred)
    results[train_name] = {
        'accuracy': accuracy,
        'classification_report': clf_report
    }

print("\n  n_estimators=200      :")
for dataset, result in results.items():
    print(f"\n  : {dataset}")
    print(f"  : {result['accuracy']:.4f}")
    print("   :")
    print(result['classification_report'])
```

```
  n_estimators=200      :

  : train_val_data_pca-9.csv
  : 0.9110
   :
              precision    recall  f1-score   support

           0       0.93      0.96      0.95       252
           1       0.72      0.57      0.64        40

    accuracy                           0.91       292
   macro avg       0.83      0.77      0.79       292
weighted avg       0.91      0.91      0.91       292


  : train_val_data_pca-7.csv
  : 0.9041
   :
              precision    recall  f1-score   support

           0       0.93      0.96      0.95       252
           1       0.70      0.53      0.60        40

    accuracy                           0.90       292
   macro avg       0.81      0.74      0.77       292
weighted avg       0.90      0.90      0.90       292


  : train_val_data_pca-5.csv
  : 0.8459
   :
              precision    recall  f1-score   support
           0       0.87      0.97      0.92       252
           1       0.22      0.05      0.08        40

    accuracy                           0.85       292
   macro avg       0.54      0.51      0.50       292
weighted avg       0.78      0.85      0.80       292


  : train_val_data.csv
  : 0.8938
   :
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       252
           1       0.65      0.50      0.56        40

    accuracy                           0.89       292
   macro avg       0.78      0.73      0.75       292
weighted avg       0.89      0.89      0.89       292
```

[33]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

processing_list = ['pca-5', 'pca-7', 'pca-9', 'no pca']
accuracy_list = [results[mapping[m]]['accuracy'] for m in processing_list]

hyper_parameter_optimze = {
    'Data processing method': processing_list,
    'Accuracy': accuracy_list
}
hyper_parameter_optimze_frame = pd.DataFrame(hyper_parameter_optimze)

colorlist = ['#de9099', '#57a1c3', '#aaa1e3', '#d988cd']

sns.barplot(x='Accuracy',
            y='Data processing method',
            data=hyper_parameter_optimze_frame,
            palette=colorlist,
            edgecolor='#DDDDDD')

plt.title("Random Forest", fontsize=14)
plt.xlabel("Accuracy", fontsize=12)
plt.ylabel("Data processing method", fontsize=12)

plt.show()
```

```
C:\Users\10940\AppData\Local\Temp\ipykernel_11468\2496371174.py:23:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='Accuracy',
```



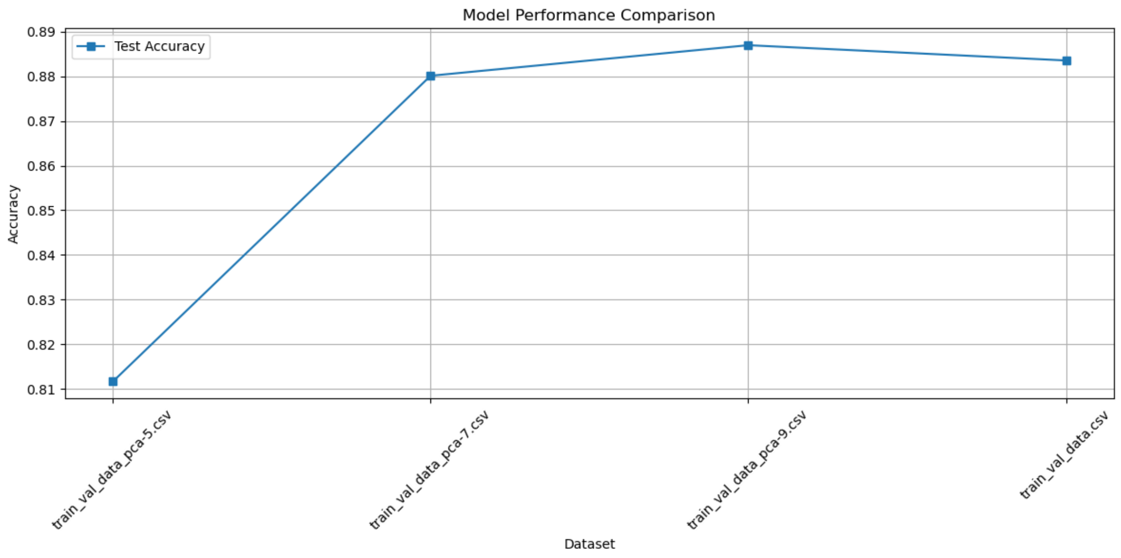
```
[ ]:
```

December 16, 2024

[2]:
```
#dataset6
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score
import matplotlib.pyplot as plt

train_files = [
    'dataset_6/train_val_data_pca-5.csv',
    'dataset_6/train_val_data_pca-7.csv',
    'dataset_6/train_val_data_pca-9.csv',
    'dataset_6/train_val_data.csv'
]
test_files = [
    'dataset_6/test_data_pca-5.csv',
    'dataset_6/test_data_pca-7.csv',
    'dataset_6/test_data_pca-9.csv',
    'dataset_6/test_data.csv'
]

#
best_params = {
    'C': 95,
    'class_weight': {0: 1, 1: 1.5},
    'gamma': 0.011,
    'kernel': 'rbf'
}

results = []
for train_file, test_file in zip(train_files, test_files):
    print(f"\n    :\nTrain: {train_file}\nTest: {test_file}")

    train_data = pd.read_csv(train_file)
    test_data = pd.read_csv(test_file)

    X_train = train_data.iloc[:, :-1]
    y_train = train_data.iloc[:, -1]
    X_test = test_data.iloc[:, :-1]
    y_test = test_data.iloc[:, -1]

    #
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    model = SVC(**best_params)
    model.fit(X_train_scaled, y_train)

    #
    y_pred = model.predict(X_test_scaled)
    test_accuracy = accuracy_score(y_test, y_pred)

    results.append({
        'file': train_file.split('/')[-1],
        'test_accuracy': test_accuracy
    })

    print(f"\n    : {test_accuracy:.4f}")
    print(classification_report(y_test, y_pred))

#
plt.figure(figsize=(12, 6))
x = range(len(results))
test_scores = [r['test_accuracy'] for r in results]
labels = [r['file'] for r in results]
plt.plot(x, test_scores, 's-', label='Test Accuracy')
plt.xticks(x, labels, rotation=45)
plt.xlabel('Dataset')
plt.ylabel('Accuracy')
plt.title('Model Performance Comparison')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
    :
Train: dataset_6/train_val_data_pca-5.csv
Test: dataset_6/test_data_pca-5.csv

   : 0.8116
              precision    recall  f1-score   support
```

```
           0       0.86      0.94      0.90       252
           1       0.00      0.00      0.00        40

    accuracy                           0.81       292
   macro avg       0.43      0.47      0.45       292
weighted avg       0.74      0.81      0.77       292


    :
Train: dataset_6/train_val_data_pca-7.csv
Test: dataset_6/test_data_pca-7.csv

   : 0.8801
              precision    recall  f1-score   support

           0       0.92      0.94      0.93       252
           1       0.57      0.50      0.53        40

    accuracy                           0.88       292
   macro avg       0.75      0.72      0.73       292
weighted avg       0.87      0.88      0.88       292


    :
Train: dataset_6/train_val_data_pca-9.csv
Test: dataset_6/test_data_pca-9.csv

   : 0.8870
              precision    recall  f1-score   support

           0       0.92      0.95      0.94       252
           1       0.61      0.50      0.55        40

    accuracy                           0.89       292
   macro avg       0.76      0.72      0.74       292
weighted avg       0.88      0.89      0.88       292


    :
Train: dataset_6/train_val_data.csv
Test: dataset_6/test_data.csv

   : 0.8836
              precision    recall  f1-score   support

           0       0.93      0.94      0.93       252
           1       0.58      0.55      0.56        40
    accuracy                           0.88       292
   macro avg       0.75      0.74      0.75       292
weighted avg       0.88      0.88      0.88       292
```


Model Performance Comparison
Test Accuracy
Accuracy
train_val_data_pca-5.csv
train_val_data_pca-7.csv
train_val_data_pca-9.csv
train_val_data.csv
Dataset