Final Project (Group 24)

Zhenghao Xiong, Jennie Lu, Jingyuan Zhu

April 30, 2023

1. Introduction

Real-life phenomenon is always influenced by multiple factors. To understand the relation of factors and phenomenon, and to predict it, we could use a more efficient method: supervised machine learning. Supervised machine learning is based on data sets given inputs and outputs, this is different from unsupervised machine learning, which only depends on inputs. The final goal of machine learning is to achieve decision-making through modeling in a short time. Models are normally categorized to be discriminative and generative, based on boundary and probability, respectively.

In this project, we analyze the red wine quality contributed by *UCI Machine Learning Repository*. The Dataset has 11 input types (fixed acidity, residual sugar, pH, etc.) which are potentially related to the output quality (Fig.1a). Besides, mutual effects of inputs remain unknown and most of them are non-linear separable (Fig.1b) and unbalanced (Fig.1c), which makes it more challenging for modeling.

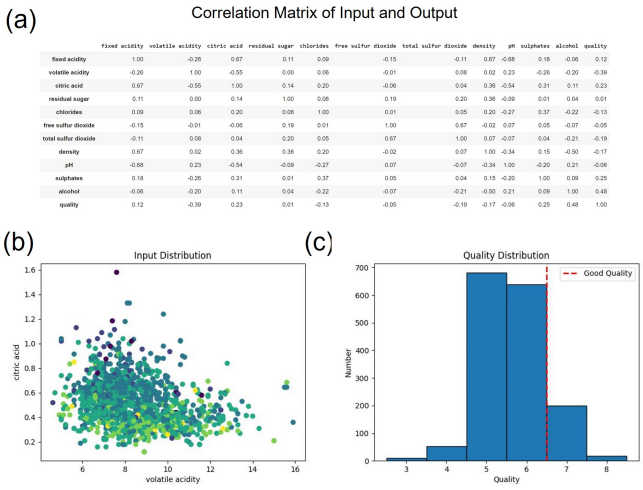

Figure 1: (a) Correlation matrix of inputs and output. (b) Non-linear separable dataset example. (c) Histogram of output distribution.

To perform a better model on red wine quality prediction, 4 different modeling methods are tested first: Random Forest, Artificial Neural Network (ANN), K-Nearest Neighbors (KNN), and Support Vector Machine (SVM). We measured and analyzed the accuracy scores for each of the models to test the model sensitivity for dataset. We then applied a model-based Binary Filter on dataset, which improved the performance of the 4 models.

2. Method

2.1 Pre-Processing

In machine learning, it is indispensable to use clean and normalized data. However, not all data requires preprocessing. To check the necessity, we first check if there exists missing data. We then test 2 preprocessing method: standard scaler and l_2 normalization supported by *sklearn.preprocessing*, on the dataset.

Missing data check shows there's no null data. The accuracy score before standard scaler and l_2 normalization is higher than that after applying them, representing dataset is clean enough. Thus, there's no need for further preprocessing.

2.2 Description of Models

For the purpose of the classification problem, we introduce four different models. The following graphs demonstrate basic structures of each model. Random Forest: It is an ensemble learning method that combines multiple decision trees to improve the accuracy and robustness of the model. We introduce a Random Forest Classifier that takes default parameter values. Specifically, we take $n\_estimators = 100$ (number of decision trees), $criterion =' gini'$ (quality of split), $max\_depth = None$ (This means

that nodes are expanded until all leaves are pure) etc. Next, for the Artificial Neural Network model, we utilize the Multi-layer Perceptron model, which has 20 hidden layers, equipped with Logistic activation function, and has max limit of iterations at 1000. The k-Nearest Neighbor (KNN) algorithm is a popular machine learning algorithm used for classification problem. It is a non-parametric algorithm that makes predictions based on the k most similar data points in the training data. We employ the K-Nearest Neighbor algorithm with k=20 by creating a KNeighborsClassifier instance and set the *n_neighbors* hyperparameter to 20. Lastly, we implement a Support Vector Machine model using 'SVC', which is the SVM model specifically used for Classification. On the top of all the default values, we changed the kernel to radial basis function (RBF). One of the main reasons of using the RBF kernel is that it can model complex, non-linear decision boundaries. Besides, it has only one hyperparameter, known as the gamma parameter, which controls the width of the kernel. This makes it relatively easy to tune the hyperparameters of the SVM.

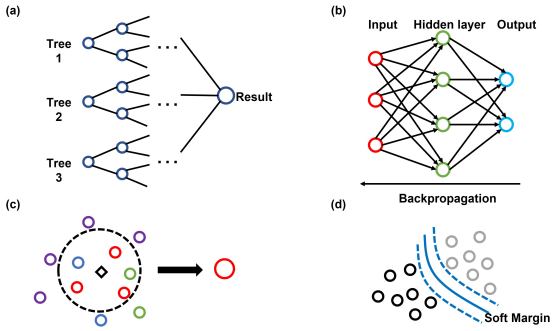

Figure 2: (a) Random Forest algorithm with 3 decision trees. (b) ANN with backward propagation training algorithm. (c) KNN result (red) of input (diamond). (d) SVM with soft margin.

2.3 Comparing Models through Accuracy Score

To find the best model, we test 4 different models and measure the performance through accuracy score, given by:

$$Accuracy = \frac{TP}{TP + NP} \tag{1}$$

where TP and NP denotes *true prediction* and *false prediction*, respectively.

Without tuning hyperparameters and applying the Binary filter introduced in the later section, we have the following accuracy results for the vanilla models:

Models	Accuracy
Random Forest	0.67
Artificial Neural Network	0.58
K-Nearest Neighbour	0.5
Support Vector Machine	0.54

Table 1: Summary of values for paired t-test.

The results indicate that the classification problem is indeed intractable since the accuracy score for three of the vanilla models are all a little higher than 0.5 (which is the score expected for random guessing) except for the Radnom Forest model, which has 0.67 accuracy score. This brings us special interest to the Random Forest model given that it has promising preliminary results for this particular data set. However, given the current ordinary results, we introduce the Binary Filter Method so as to improve accuracy for these models.

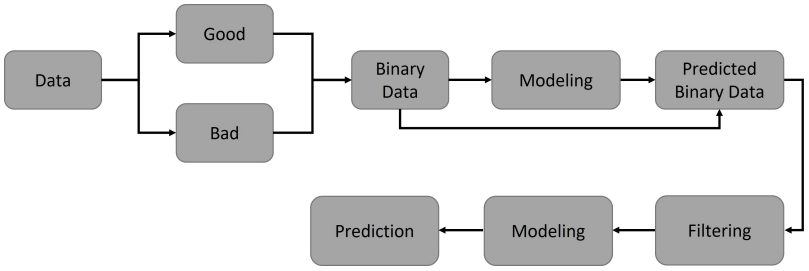

Figure 3: Scheme of model-based binary filter modeling process

2.4 Model-based Binary Filter

Red wine quality level ranges from 3 to 8, indicating quality prediction is a multi-class problem. However, quality can be classified as good for $quality > 6.5$ and bad for others (Fig.3). Factors affecting quality may have great difference in these 2 classes. We thus present a binary filter to improve the performance of the model.

Focusing on *bad* and *good*, models can achieve prediction with high accuracy. To implement binary filter, we first test model performance on binary dataset. Model with highest accuracy is then used to predict the whole dataset, generating a binary predicted output. By comparing the result with true output, those misclassified data is dropped as outlier. Since performance in binary data has very high accuracy, those dropped data will not affect model reliability much. *good* and *bad* data is then split into 2 datasets for training (Fig.3).

To evaluate the performance of binary model, accuracy is rewritten as:

$$Accuracy_{binary} = \frac{TP_0 \cdot N_0 + TP_1 \cdot N_1}{N_0 + N_1} \tag{2}$$

where TP_0, TP_1 denotes true predictions of 2 models. N_0, N_1 is the test set size of each dataset. Since we use the same test size for both datasets, N_0, N_1 can also be the each data size itself.

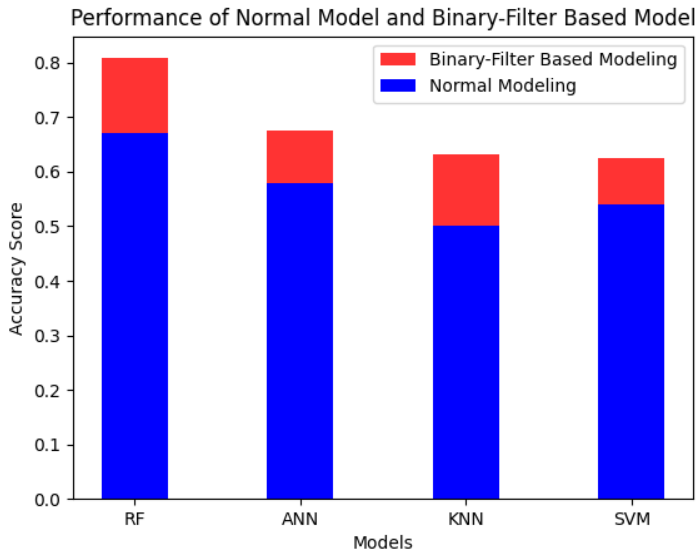

Figure 4: Comparison of non-filtered modeling and binary-filter based accuracy

3. Results

All the 4 models can classify raw data to good quality and bad quality with a high accuracy (Random Forest modeling achieves best at 92%). Performance on each binary classification is better than classifying the data directly, with 5% to 15% on bad quality and 40% to 70% on good quality (Table 1). Binary-filter based models improve the performance on 4 models over 10% (Fig.4). Binary accuracy scores on good quality data among 4 models are same, which indicates good quality data is supposed to be linear separable.

Performance of Random Forest algorithm is the best in all modeling processes (non-filtered modeling, binary classification, and binary-filter based modeling), indicating it is the most suitable method among 4 tested models.

Accuracy Score Performance				
Model	RF	ANN	KNN	SVM
Non-Filtered Model	0.67	0.5	0.58	0.54
Binary Classification	0.92	0.9	0.88	0.88
Bad Quality	0.79	0.64	0.59	0.58
Good Quality	0.94	0.94	0.94	0.94
Binary-Filter Based Model	0.82	0.68	0.63	0.62

Table 2: Performance under 4 tested models of each process.

4. Conclusion

Through observing the input feature scatter plot and the quality distribution histogram, we expected that the Red Wine data set entails a complicated classification problem since the patter of the underlying data set is non-linear. This is proven true as the accuracy scores of Artificial Neural Network, K-Nearest Neighbor, and Support Vector Machine is very low. However, we have discovered that all four model have significantly higher accuracy scores if we use binary classification and divide the data set into two groups. Because of this, we applied the binary filter and introduced a new performance evaluation scheme. The binary-filter based models' accuracy score has increased. It has a 10% improvement from the original model. The Random Forest Model seems to be the winning model for this data set, with a 82% accuracy score.

Work Distribution: Zhenghao Xiong: Introduction, Pre-Processing, Accuracy Evaluation, Binary-filter Methods Jennie Lu: Introduction, Conclusion, Appendix Jingyuan Zhu: Description of Models, Comparing Vanilla Models, Conclusion

Appendix

The notebook file and running outputs can be found at this Github repository: https://github.com/Jingyuan-zhu/Pattern-Classification-Final_Project-.

```
"""# Initialize"""

from google.colab import files
uploaded = files.upload()

import numpy as np
import pandas as pd
import io
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

np.random.seed(2)
data = pd.read_csv('winequality-red.csv', delimiter=';')
y = data.quality.values  # get the target
X = data.drop('quality', axis=1).values  # get the input
data.corr().round(2)
data
```

```
data.isnull().sum()  # check NaN data

"""# Input Distribution"""

plt.figure()
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.xlabel(data.columns[1])
plt.ylabel(data.columns[2])
plt.title('Input Distribution')
plt.savefig('input.jpg')

"""# Quality Distribution"""

plt.figure()
plt.hist(y, bins=[2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5], edgecolor='black')
plt.axvline(6.5, color='red', ls='—',lw=2, label='Good Quality')
plt.title('Quality Distribution')
plt.ylabel('Number')
plt.xlabel('Quality')
plt.legend()
plt.savefig('quality dist.jpg')

"""# Model Shortcut"""

def model(X, y, prediction=False):
  X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75)
```

```
rf = RandomForestClassifier()  # Randon Forest
ann = MLPClassifier(activation='logistic', hidden_layer_sizes=20,
                    max_iter=1000)  # ANN
knn = KNeighborsClassifier(n_neighbors=20)  # KNN
svm = SVC(kernel='rbf')  # Support Vector Machine
# predict
y_rf = rf.fit(X_train, y_train).predict(X_test)
y_ann = ann.fit(X_train, y_train).predict(X_test)
y_knn = knn.fit(X_train, y_train).predict(X_test)
y_svm = svm.fit(X_train, y_train).predict(X_test)
# accuracy
accuracy_rf = accuracy_score(y_rf, y_test)  # RF accuracy
accuracy_ann = accuracy_score(y_ann, y_test)  # ANN accuracy
accuracy_knn = accuracy_score(y_knn, y_test)  # KNN accuracy
accuracy_svm = accuracy_score(y_svm, y_test)  # SVM accuracy
Accuracy = [accuracy_rf, accuracy_ann, accuracy_knn, accuracy_svm]
Accuracy = np.round(Accuracy, decimals=2)
# print result
print('RF Accuracy:', Accuracy[0])
print('ANN Accuracy:', Accuracy[1])
print('KNN Accuracy:', Accuracy[2])
print('SVM Accuracy:', Accuracy[3])
# get predicted y_binary
if prediction:
 y_pre = rf.fit(X_train, y_train).predict(X)
 return Accuracy, y_pre
return Accuracy
```

```
"""# Binary Accuracy Shortcut"""

def accuracy(size0, size1, accuracy0, accuracy1):
  group_name = ['RF', 'ANN', 'KNN', 'SVM']
  accuracy0, accuracy1 = np.array(accuracy0), np.array(accuracy1)
  Accuracy = (accuracy0 * size0 + accuracy1 * size1) / (size0 + size1)
  for i in range(Accuracy.size):
    print(group_name[i], 'Total Accuracy:', np.round(Accuracy[i], decimals=2))
  return Accuracy

"""# Modeling Directly"""

Accuracy_d = model(X, y)

"""# Binary Classification"""

# Binary Modeling
y_binary = np.sign(y-6.5)  # separate good and bad quality
Accuracy, y_pre = model(X, y_binary, prediction=True)

"""# Binary Filter"""

ind_bad = y_binary + y_pre == -2
ind_good = y_binary + y_pre == 2
X0, y0 = X[ind_bad, :], y[ind_bad]  # bad quality
X1, y1 = X[ind_good, :], y[ind_good]  # good quality
```

```
"""# Classification on Bad Quality"""

# bad-quality accuracy
Accuracy0 = model(X0, y0)

"""# Classification on Good Quality"""

# good quality accuracy
Accuracy1 = model(X1, y1)

"""# Total Accuracy"""

# total accuracy
Accuracy_binary = accuracy(y0.size, y1.size, Accuracy0, Accuracy1)

"""# Compare Result of Normal Model and Binary Filtered Model"""

label = ['RF', 'ANN', 'KNN', 'SVM']  # model label
plt.figure()
plt.bar(label, height = Accuracy_binary, alpha=0.8,
        label = 'Binary-Filter Based Modeling', color = 'red', width = 0.4)
plt.bar(label, height = Accuracy_d, label = 'Normal Modeling', color = 'blue',
        width = 0.4)
plt.xlabel('Models')
plt.ylabel('Accuracy Score')
plt.title('Performance of Normal Model and Binary-Filter Based Model')
```

```
plt.legend()
plt.savefig('Fig4')
```