Final Project: Predict Apple Quality

Ziwei Su, Wenhao Gu

1 Introduction

Evaluation and prediction of fruit quality are critical for ensuring a steady supply of high-quality fruits to meet market demands. The Apple Quality dataset, available on Kaggle Apple Quality, presents an interesting and valuable resource for predicting the quality of apples based on diverse characteristics. The primary objective of this project is to conduct a comparative analysis of various predictive models to achieve high predictive accuracy in determining apple quality while concurrently evaluating the significance of various apple attributes in predicting apple quality.

2 Data Description

The dataset comprises 4,000 rows and 9 columns, with each row corresponding to the distinct attributes of an individual apple. These attributes include:

- **A_id**: Unique identifier for each fruit.
- **Size**: Size of the fruit.
- **Weight**: Weight of the fruit.
- **Sweetness**: Degree of sweetness of the fruit.
- **Crunchiness**: Texture indicating the crunchiness of the fruit.
- **Juiciness**: Level of juiciness of the fruit.
- **Ripeness**: Stage of ripeness of the fruit.
- **Acidity**: Acidity level of the fruit.
- **Quality**: Overall quality of the fruit. It takes value in $\{\text{good}, \text{bad}\}$.

The following table gives the first five rows of the Apple Quality dataset.

A_id	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	Acidity	Quality
0	-3.970049	-2.512336	5.346330	-1.012009	1.844900	0.329840	-0.491590	good
1	-1.195217	-2.839257	3.664059	1.588232	0.853286	0.867530	-0.722809	good
2	-0.292024	-1.351282	-1.738429	-0.342616	2.838636	-0.038033	2.621636	bad
3	-0.657196	-2.271627	1.324874	-0.097875	3.637970	-3.413761	0.790723	good
4	1.364217	-1.296612	-0.384658	-0.553006	3.030874	-1.303849	0.501984	good

Table 1: Apple Quality Dataset Sample

Figure 1 summarizes the dataset. We choose Size, Weight, Sweetness, Crunchiness, Juiciness, Ripeness, and Acidity as the predictors, and Quality is the binary response. We exclude A_id because the identifier is irrelevant to our analysis. We transform Acidity to float, normalize all the predictors, and transform the binary response to $\{0, 1\}$. Figure 2 displays the correlation coefficients among the 7 predictors, showing little collinearity among them. This can also be observed from the scatter matrix in Figure 3.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Size         4000 non-null   float64
 1   Weight       4000 non-null   float64
 2   Sweetness    4000 non-null   float64
 3   Crunchiness  4000 non-null   float64
 4   Juiciness    4000 non-null   float64
 5   Ripeness     4000 non-null   float64
 6   Acidity      4000 non-null   object
 7   Quality      4000 non-null   object
dtypes: float64(6), object(2)
memory usage: 250.1+ KB
```

Figure 1: Summary of the Apple Quality Dataset

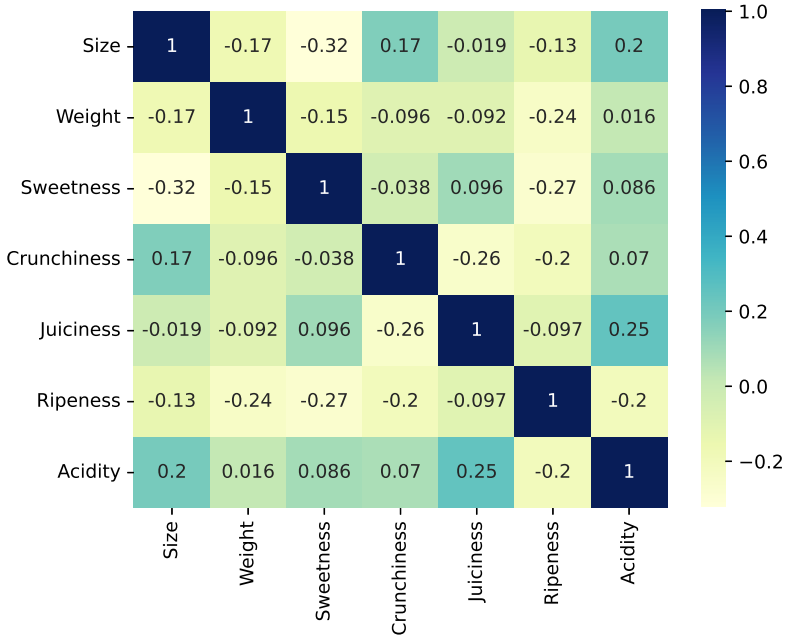

Figure 2: Correlations of the predictors

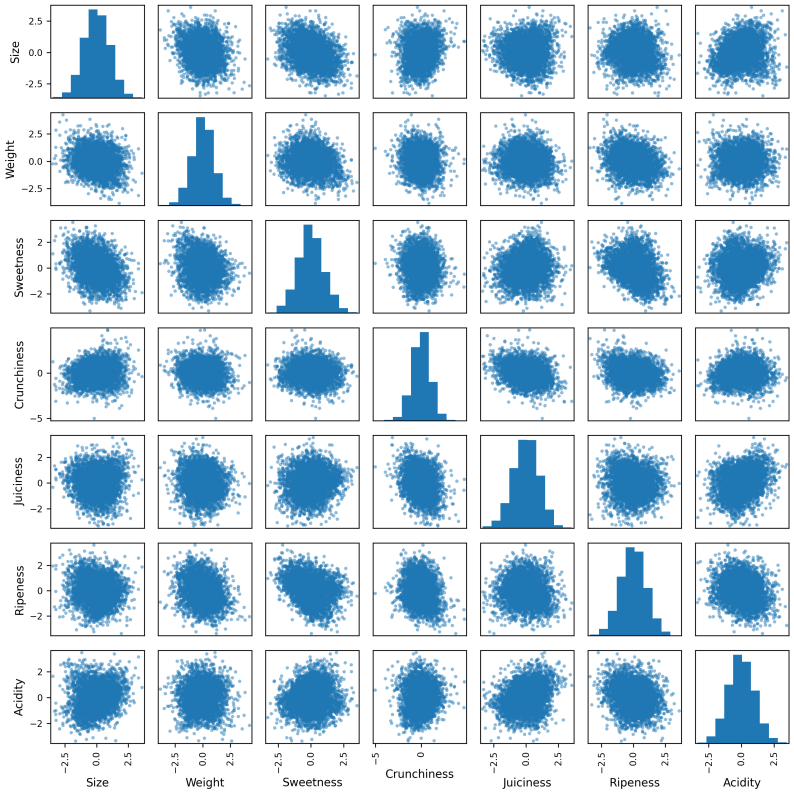

Figure 3: Scatter matrix of the predictors

3 Model Fitting and Analysis

In this section, we will be deploying the following eight models to the Apple Quality dataset for classification: Logistic Regression (LogR), Naive Bayes (NB), Linear Discriminant Analysis (LDA), Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Neural Network (NN), Classification Tree (CT), Light Gradient Boosting Machine (LGBM), and Random Forest (RF). We measure the predictive power of each model with the mean cross-validation (CV) error with multiple replications of 10-fold CV, summarized in Table 2. In general, we use the permutation-based variable importance

Table 2: Mean CV accuracy of models.

Model	CV accuracy
Logistic Regression (LogR)	0.748
Naive Bayes (NB)	0.749
Linear Discriminant Analysis (LDA)	0.748
Support Vector Machine (SVM)	0.915
K-Nearest Neighbors (KNN)	0.900
Neural Network (NN)	0.946
Classification Tree (CT)	0.819
Light Gradient Boosting Machine (LGBM)	0.900
Random Forest (RF)	0.891

measure (VIM) to quantify the importance of the predictors. We use other approaches for the predictor importance such as the Accumulated Local Effects (ALE) plot for the logistic regression model and the impurity-based VIM for tree models.

3.1 Logistic Regression (LogR)

In this section, we use the logistic regression model with *sklearn.linear_model.LogisticRegression*. The hyperparameters to tune are C, *penalty*, and $l1\_ratio$, where C is the inverse of regularization strength, *penalty* is the type of regularization penalty, and $l1\_ratio$ is a mixing parameter taking values between 0 and 1. Possible regularization methods include $l1$ (LASSO), $l2$ (Ridge), and elastic net, a mixture of $l1$ and $l2$ regularization. $l1\_ratio = 0$ is equivalent to using $l2$ penalty, while setting $l1\_ratio = 1$ is equivalent to using $l1$ penalty. For $0 < l1\_ratio < 1$, the penalty is a combination of $l1$ and $l2$. We perform grid search with 10-fold cross-validation repeated 10 times on the following hyperparameter grid:

$$C \in \{0.01, 0.05, 0.1, 0.5, 1, 5, 10\}, penalty \in \{l1, l2, elasticnet, none\}, l1\_ratio \in \{0.1, 0.3, 0.5, 0.7, 0.9\}.$$

The best hyperparameter configuration is $C = 0.05$, $l1\_ratio = 0.7$, $penalty = elasticnet$, and the corresponding model has a CV mean accuracy of around 0.748.
We consider the importance of predictors via the coefficients of each predictor in the model, the permutation VIM, and the ALE plot against the log odds. The feature coefficients are as follows:

- Size: The coefficient for size is 1.1277.
- Sweetness: The coefficient for sweetness is 0.9920.
- Juiciness: The coefficient for juiciness is 0.7664.
- Acidity: The coefficient for acidity is -0.5371.
- Weight: The coefficient for weight is 0.3507.
- Ripeness: The coefficient for ripeness is -0.2454.
- Crunchiness: The coefficient for crunchiness is 0.0096.

The permutation VIM is given in Figure 4, and the ALE plot is given in Figure 5. We can tell from the coefficients, the permutation VIM, and the ALE that Size appears to be the most important predictor, followed by Sweetness, and Crunchiness appears to be the least important.

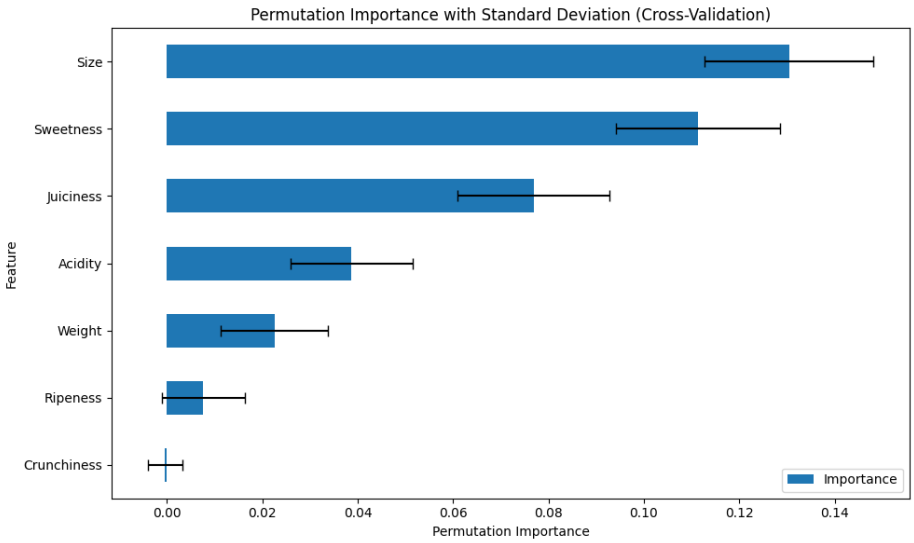

Figure 4: Permutation VIM of the best LogR model

3.2 Naive Bayes (NB)

In this section, we use the Naive Bayes model with $sklearn.naive\_bayes.GaussianNB$. The only hyperparameter to tune is the amount of smoothing ($var\_smoothing$), which is added to the variance of each feature during fitting to alleviate overfitting. We perform grid search with 10-fold cross-validation on the following hyperparameter grid:

$$var\_smoothing \in \{1e-2, 5e-2, 1e-1, 5e-1, 1, 5, 10\}.$$

The best hyperparameter value is $var\_smoothing = 0.01$, and the corresponding model achieves a CV mean accuracy of around 0.749. We consider the importance of predictors via the permutation VIM, given in Figure 6. According to the permutation VIM, the order of predictor importance from most to least is: Juiciness, Size, Sweetness, Ripeness, Weight, Crunchiness, and Acidity. The first six predictors vary little in their VIM scores, while Acidity appears to be very unimportant in comparison.

3.3 Linear Discriminant Analysis (LDA)

In this section, we use the Linear Discriminant Analysis model with $sklearn.discriminant\_analysis$. There are no hyperparameters to tune, as when we set the parameter $solver =' lsqr'$ (least squares solution), the shrinkage parameter can be automatically selected using the Ledoit-Wolf lemma. The model achieves a CV mean accuracy of around 0.7479. We consider the importance of predictors via the permutation VIM, given in Figure 7. According to the permutation VIM, the most important predictors in descending order are: Size, Sweetness, and Juiciness, while the least important predictor is Crunchiness.

3.4 Support Vector Machine (SVM)

In this section we use the support vector machine (SVM) algorithm with $sklearn.svm.SVC$. The parameters are C and $kernel$, where C is the regularization parameter and $kernel$ is the kernel for the SVM classifier. But we choose $kernel =' rbf'$, meaning that we are using Guassian kernel, because $'rbf'$ is efficient to compute. we use K-fold cross validation to do a thee stage grid search over C. First we search

$$C \in \{0.1, 1, 10, 50, 100, 1000\}$$

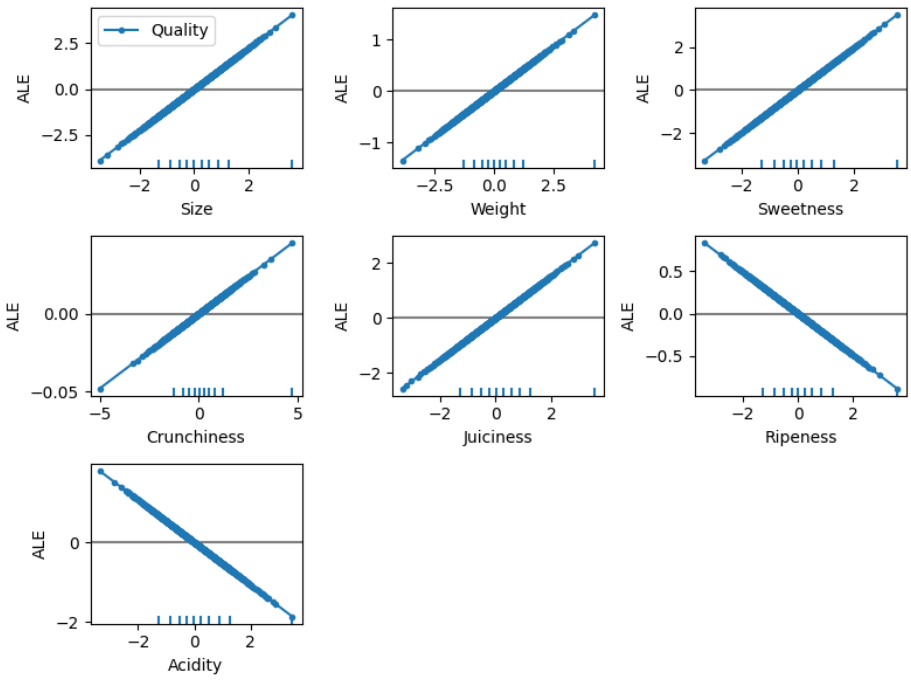

Figure 5: ALE of the best LogR model

and found that the best C in this grid is 50. We then search

$$C \in \{10, 20, 30, ..., 90, 100\}$$

and found that the best parameter is $C = 40$. Finally, we search

$$C \in \{30, 31, 32, ..., 49, 50\}$$

and the best parameter is $C = 41$ with the best CV accuracy being 0.9151. For the model with the selected best parameter, we calculate the CV permutation VIM importance of the predictors and visualize them in 8.
From the permutation VIM importance plot, the most important three predictors are Ripeness, Sweetness and Size, and the least important predictor is Crunchiness.

3.5 K-nearest Neighbors (KNN)

In this section we use the K-Nearest Neighbors (KNN) algorithm to fit the prediction model. The parameter for the distance-based KNN classifier is $n\_neighbors$, and we use K-fold cross validation to do a grid search over $n\_neighbors \in \{1, 2, ..., 30\}$. The resulting cross validation accuracy versus $n\_neighbors$ is shown in Figure 9.
The best parameter is $n\_neighbors = 11$, and the CV accuracy is 0.90045. For the model with the selected best parameter, we calculate the CV permutation VIM importance of the predictors, which are shown in Figure 10.
From the permutation VIM importance plot, the most important three predictors are Ripeness, Size and Sweetness, and the least important two predictors are Crunchiness and Weight.

3.6 Neural Network (NN)

In this section, we use a neural network model with $sklearn.neural\_network.MLPClassifier$. The activation function is the logistic sigmoid function. The hyperparameters to tune are the number of

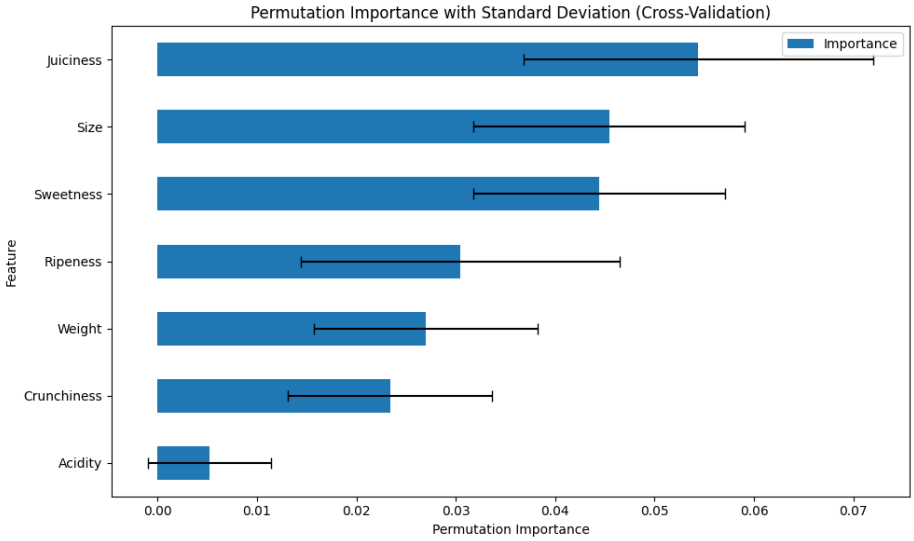

Figure 6: Permutation VIM of the best Naive Bayes model

neurons in the hidden layer (*hidden_layer_sizes*), the $l2$ regularization parameter α, and the initial learning rate (*learning_rate_init*). We perform a grid search with 10-fold cross-validation repeated 1 time on the following hyperparameter grid:

$$hidden\_layer\_sizes \in \{(50,),(100,)\}, \alpha \in \{0.0001, 0.001, 0.01\}, learning\_rate\_init \in \{0.01, 0.1\}.$$

The neural network is very costly to tune with cross-validation. Due to time constraints, only 1 replication of 10-fold cross-validation was performed. The best hyperparameter configuration is $\alpha = 0.01$, $hidden\_layer\_sizes = (50,)$, $learning\_rate\_init = 0.01$, and the corresponding model has a CV mean accuracy of around 0.946. We consider the importance of predictors via the permutation VIM, given in Figure 11. The permutation VIM shows that the most important predictor is Ripeness, followed closely by Sweetness and Size. The next tier of predictors, in order of decreasing importance, are Juiciness, Acidity, Weight and Crunchiness, which differ little in importance from each other. Overall, the difference in predictor importance for all predictors is not very large.

3.7 Classification Tree (CT)

In this section, we use the classification tree model $sklearn.tree.DecisionTreeClassifier$. The hyperparameters are the maximum depth of the tree (*max_depth*), the minimum number of samples required to be at a leaf node (*min_samples_split*), the minimum number of samples required to be at a leaf node (*min_samples_leaf*), and the complexity parameter used for Minimal Cost-Complexity Pruning (*ccp_alpha*). We perform a four stage grid search with 10-fold cross-validation repeated. We first search over

$$\begin{aligned} max\_depth &\in \{5, 10, 15, 20\} \\ min\_samples\_split &\in \{2, 3, 5, 10\} \\ min\_samples\_leaf &\in \{2, 3, 5, 10\} \\ ccp\_alpha &\in \{0.0, 0.001, 0.01, 0.1\} \end{aligned}$$

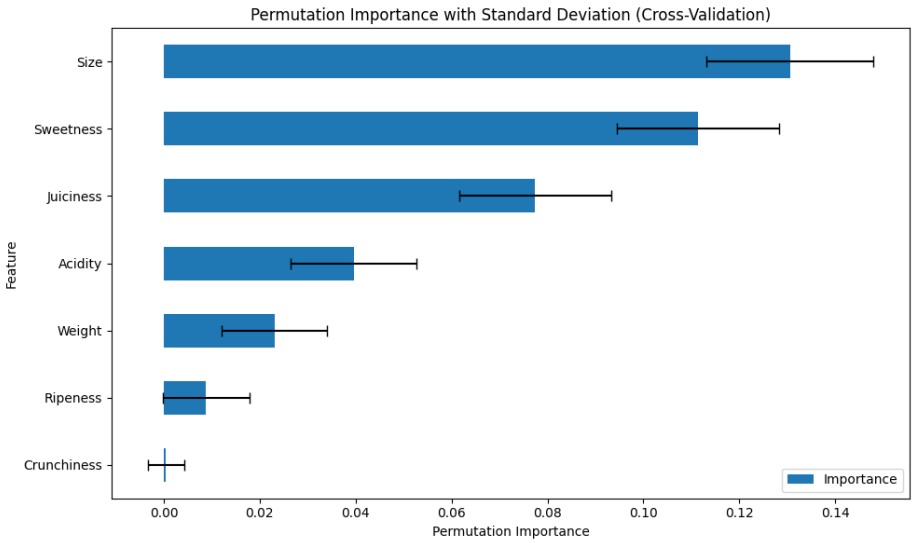

Figure 7: Permutation VIM of the LDA model

The best hyperparameter for this set of parameters is $max\_depth = 15, min_samples_split = 5, min\_samples\_leaf = 2, ccp\_alpha = 0.001$. Next we search over

$$
\begin{aligned}
&max\_depth \in \{13, 15, 17, 19\} \\
&min\_samples\_split \in \{4, 5, 7, 9\} \\
&min\_samples\_leaf \in \{2, 3, 4\} \\
&ccp\_alpha \in \{0.0, 0.0001, 0.0005, 0.001\}
\end{aligned}
$$

and the best parameter in the 2nd search is $max\_depth = 15, min_samples_split = 7, min\_samples\_leaf = 4, ccp\_alpha = 0.0005$. In the 3rd search we examine

$$
\begin{aligned}
&max\_depth \in \{14, 15, 16\} \\
&min\_samples\_split \in \{6, 7, 8\} \\
&min\_samples\_leaf \in \{3, 4\} \\
&ccp\_alpha \in \{0.0002, 0.0003, 0.0005, 0.0007, 0.0009\}
\end{aligned}
$$

and the best parameter in the 3rd search is $max\_depth = 16, min_samples_split = 8, min\_samples\_leaf = 4, ccp\_alpha = 0.0007$. In the last search, we test

$$
\begin{aligned}
&max\_depth \in \{15, 16, 17\} \\
&min\_samples\_split \in \{7, 8, 9\} \\
&min\_samples\_leaf \in \{3, 4, 5\} \\
&ccp\_alpha \in \{0.0006, 0.0007, 0.0008\}
\end{aligned}
$$

and the best model is $max\_depth = 15, min_samples_split = 7, min\_samples\_leaf = 5, ccp\_alpha = 0.0007$. The corresponding best CV accuracy is 0.8195. We show the feature importance generate by the $sklearn.tree.DecisionTreeClassifier.feature\_importances\_$ attribute in Figure 12
this shows that the most important predictor is Ripeness, and the least important predictor is Crunchiness. We next examine the CV permutation in Figure 13
this shows that the most important predictor is Ripeness, and the least two important predictors are Crunchiness and Weight, which is the same as in the results shown in the $feature\_importance\_$ function. However, the importance ranking of the other predictors are not the same.

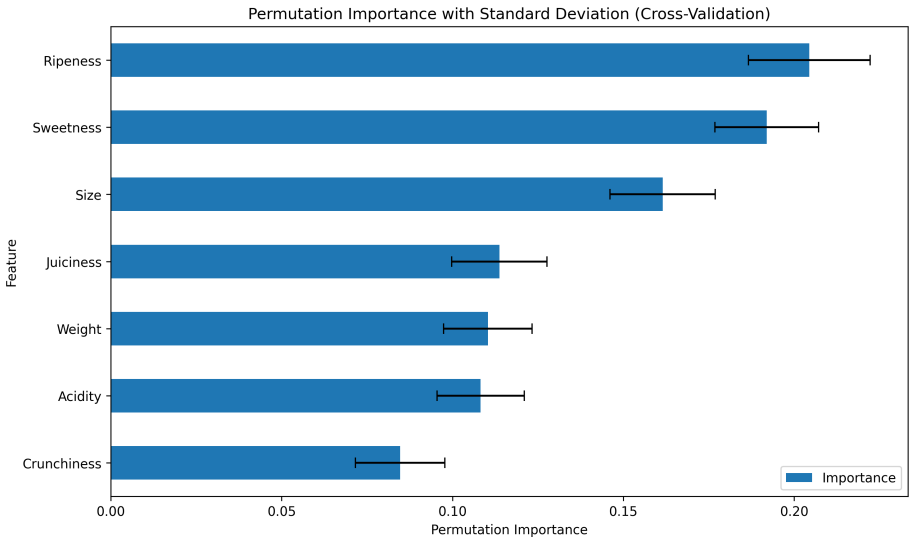

Figure 8: Permutation VIM of the best KNN model

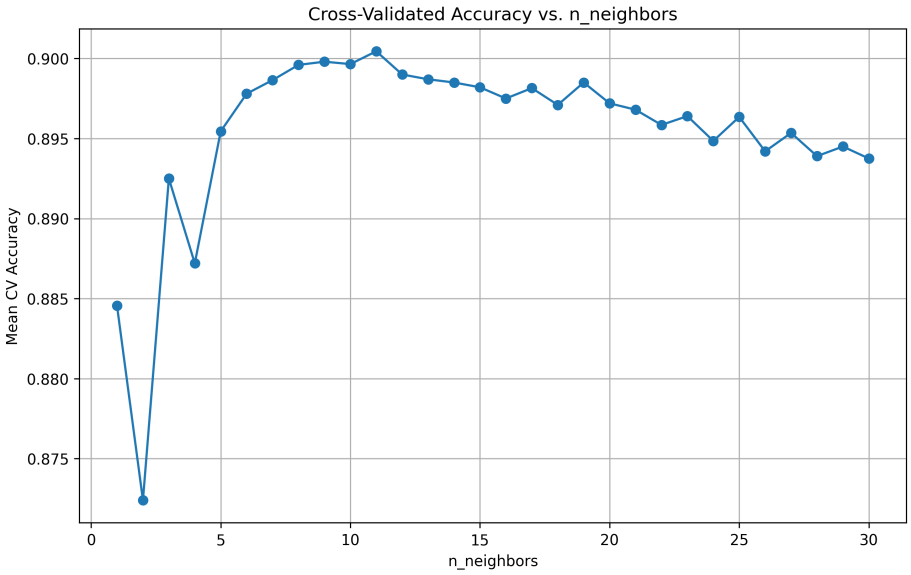

Figure 9: CV accuracy v.s. *n_neighbors*

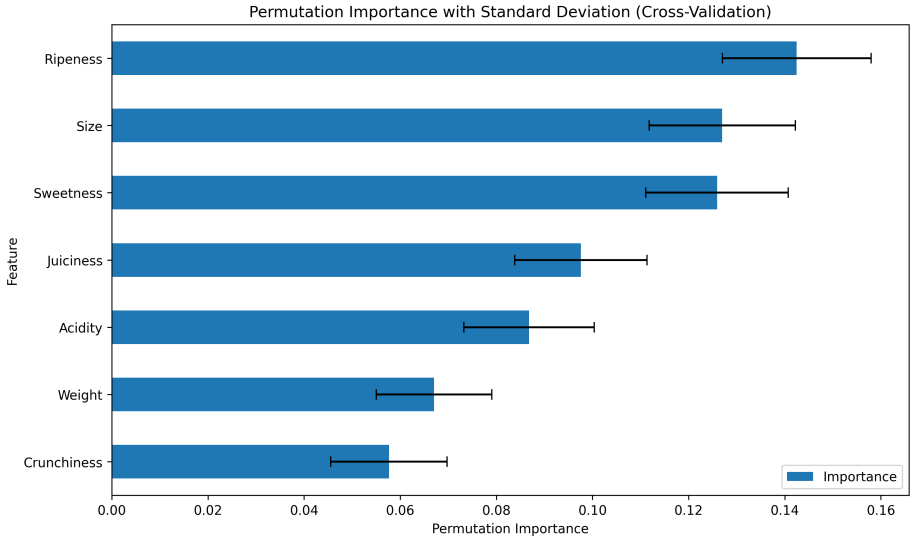

Figure 10: Permutation VIM of the best KNN model

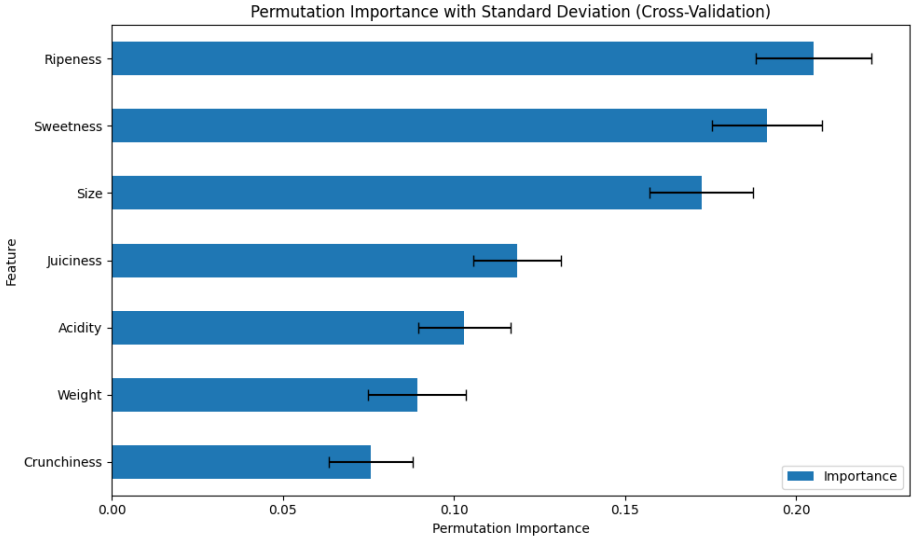

Figure 11: Permutation VIM of the best neural network model

```
Ranked Features by Importance:
1. Ripeness: 0.2231
2. Sweetness: 0.1715
3. Size: 0.1622
4. Acidity: 0.1513
5. Juiciness: 0.1398
6. Weight: 0.0959
7. Crunchiness: 0.0563
```

Figure 12: Feature importance of the best classification tree

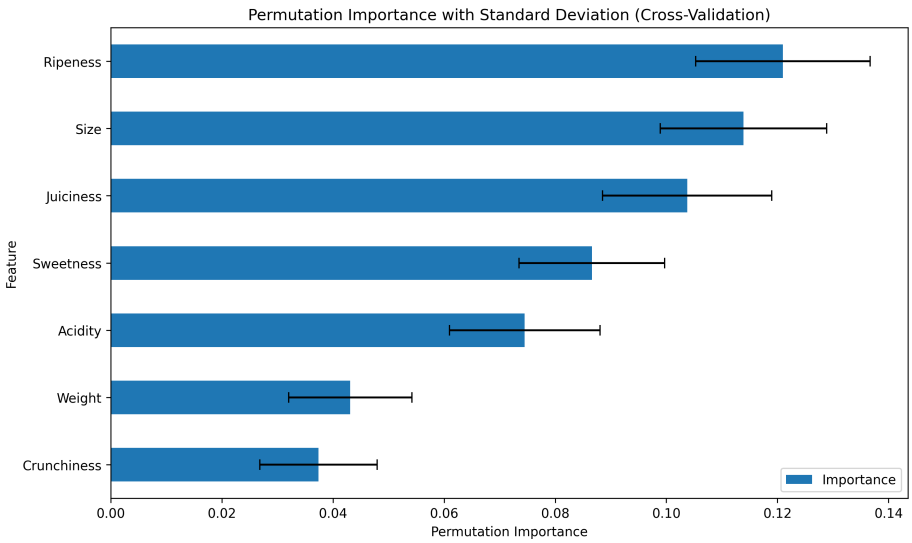

Figure 13: Permutation VIM of the best classification tree

3.8 Light Gradient Boosting Machine (LGBM)

In this section, we use the Light Gradient Boosting Machine $lightgbm.LGBMClassifier$. The hyperparameters are $learning\_rate$, $n\_estimators$, max_depth and num_leaves. We do a four stage grid search with 10-fold cross-validation repeated. We first search over

$$\begin{aligned} &learning\_rate \in \{0.01, 0.1, 0.5\} \\ &n\_estimators \in \{200, 300\} \\ &max\_depth \in \{10, 20, 30\} \\ &num\_leaves \in \{15, 31, 63\} \end{aligned}$$

The best hyperparameter for this set of parameters is $learning\_rate = 0.1, n\_estimators = 300, max\_depth = 30, num\_leaves = 63$. Next we search over

$$\begin{aligned} &learning\_rate \in \{0.05, 0.1, 0.2\} \\ &n\_estimators \in \{300, 400\} \\ &max\_depth \in \{30, 40\} \\ &num\_leaves \in \{63, 96\} \end{aligned}$$

and the best parameter in the 2nd search is $learning\_rate = 0.2, n\_estimators = 300, max\_depth = 30, num\_leaves = 96$. In the 3rd search we examine

$$\begin{aligned} &learning\_rate \in \{0.18, 0.2, 0.22\} \\ &n\_estimators \in \{300, 400\} \\ &max\_depth \in \{30\} \\ &num\_leaves \in \{96\} \end{aligned}$$

and the best parameter in the 3rd search is $learning\_rate = 0.22, n_estimators = 400, max_depth = 30, num_leaves = 96$. In the last search, we test

$$\begin{aligned} &learning\_rate \in \{0.22, 0.24, 0.26, 0.28, 0.3\} \\ &n\_estimators \in \{300, 400, 500\} \\ &max\_depth \in \{30\} \\ &num\_leaves \in \{96\} \end{aligned}$$

and the best model is $learning\_rate = 0.22, n\_estimators = 500, max\_depth = 30, num\_leaves = 96$. The corresponding best CV accuracy is 0.9003. We show the feature importance generate by the $lightgbm.plot\_importance$ function in Figure 14
this shows that the most important predictor is Ripeness, and the least important predictor is Crunchiness. We next examine the CV permutation in Figure 15
this shows that the most three important predictors are Ripeness, Size and Sweetness, and the least two important predictors are Crunchiness and Weight. However, the importance ranking of the other predictors are not the same.

3.9 Random Forest (RF)

In this section, we use the random forest model with $sklearn.ensemble.RandomForestClassifier$. The hyperparameters to tune are the number of trees ($n\_estimators$), the maximum depth of the tree (max_depth), the minimum number of samples required to split an internal node ($min\_samples\_split$), and the minimum number of samples required to be at a leaf node ($min\_samples\_leaf$). We perform a grid search with 10-fold cross-validation repeated 1 time on the following hyperparameter grid:

$$n\_estimators \in \{250, 300, 350\}, max\_depth \in \{20, 25, 30\},$$

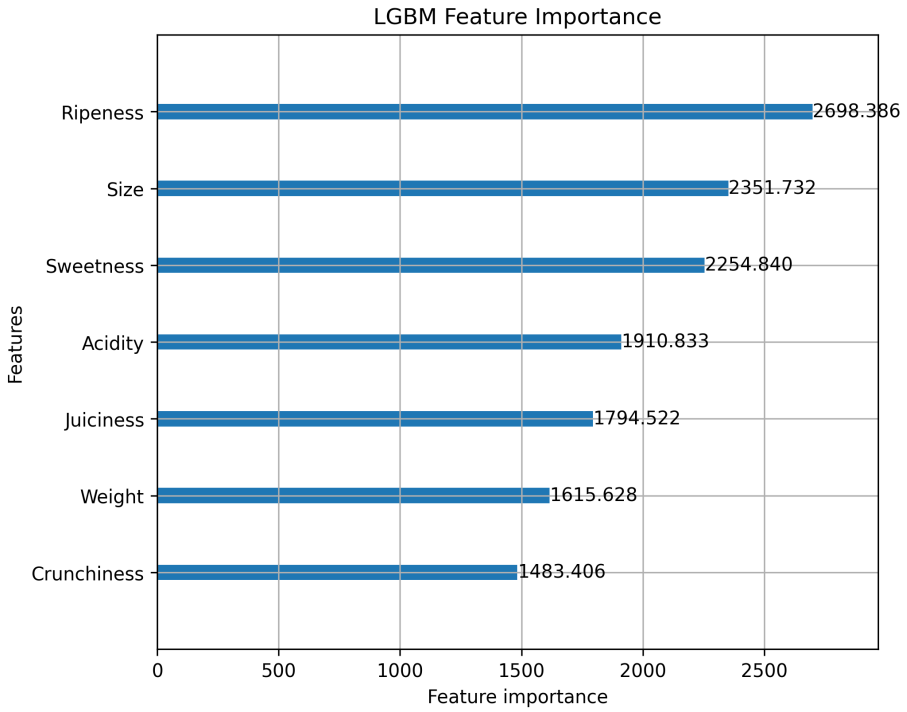

Figure 14: Feature importance of the best classification tree

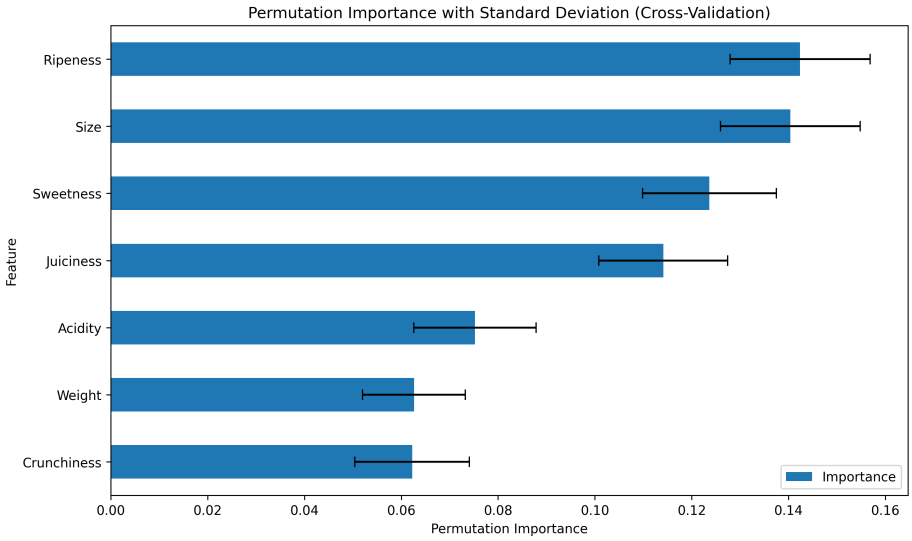

Figure 15: Permutation VIM of the best classification tree

$$min\_samples\_split \in \{2, 5, 10\}, min\_samples\_leaf \in \{1, 2, 5\}.$$

The best hyperparameter configuration is $max\_depth = 25$, $min\_samples\_leaf = 1$, $min\_samples\_split = 2$, $n\_estimators = 300$, and the corresponding model has a CV mean accuracy of around 0.891. The predictor importance is given by both the impurity-based measure (in Figure 16) and the permutation VIM (in Figure 17). Both measures agree that Ripeness, Size, Juciness are the three most

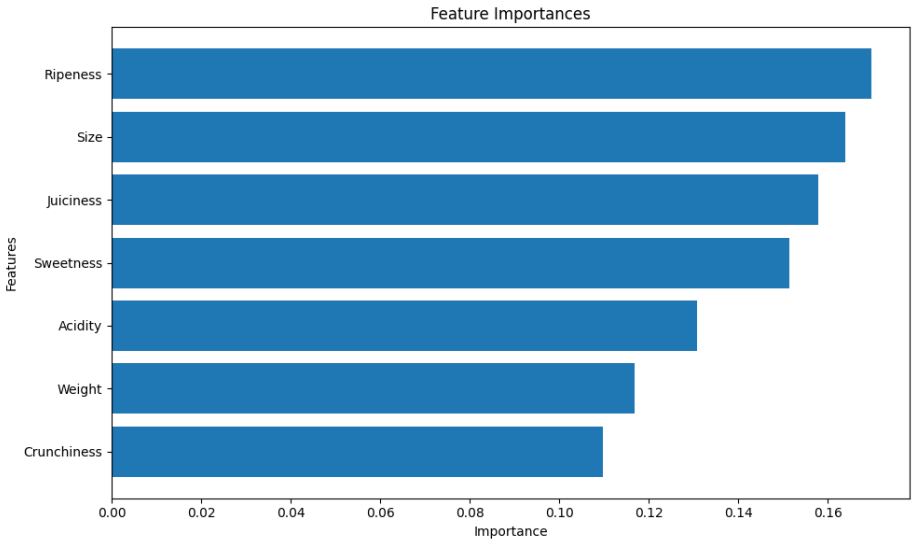

Figure 16: Impurity-based predictor importance of the best random forest model

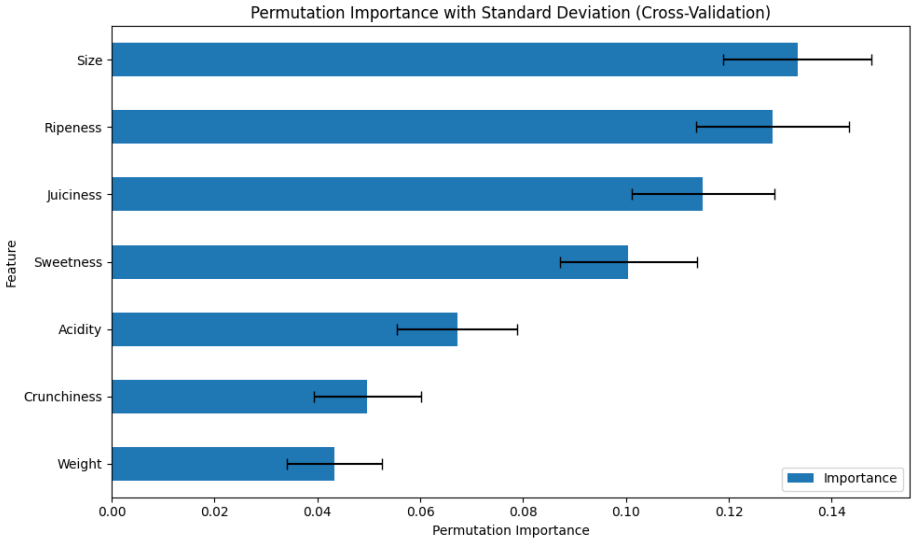

Figure 17: Permutation VIM of the best random forest model

important predictors, and Acidity, Crunchiness and Weight are the three least important predictors, but the difference between importance measures for different predictors seem small.

4 Conclusion

From Table 2, we conclude that the best model in terms of predictive power is Neural Network (NN) with a CV accuracy of around 0.946, followed by Support Vector Machine (SVM), Light Gradient Boosting Tree (LGBM), K-Nearest Neighbors (KNN) and Random Forest (RF), all with accuracy around 0.9. Simpler linear models such as Logistic Regression (LR), Naive Bayes (NB), and Linear Discriminant Analysis (LDA) only achieve CV accuracy of around 0.75.
The superior performance of NN and the relatively bad performance of linear models are expected, but the relatively good performance of KNN is unexpected for us. We speculate that the relatively good performance of KNN is due to the small number of predictors (7) and the small size of training data (4000). KNN is suitable for low-dimensional feature space due to less sparsity and a small training set due to its locality. Therefore, we recommend using NN for the best predictive power and KNN for a good trade-off between predictive performance and model interpretability.
In terms of the feature importance, different models yield different rankings. Ripeness and Size appear as the top two predictors, rank consistently among the top 3 across different models. Conversely, Crunchiness and Weights are at the lower end of the ranking, frequently appearing in the bottom 2. However, the relative rankings of the remaining predictors remain unclear to us at this point.

A Code

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

# data overview
df = pd.read_csv("apple_quality.csv")
df = df.loc[0:df.shape[0]-2,:]
df.head()

# remove id
df.drop('A_id', axis=1, inplace=True)
df.info()

# data preprocessing
df['Acidity'] = df['Acidity'].astype(float) # convert to float

# data normalization
# get the number of columns in the dataframe
num_columns = df.shape[1]

# create scaler
scaler = StandardScaler()

# select all columns except the last one
columns_to_normalize = df.columns[:num_columns-1]
print(columns_to_normalize)
# normalize the selected columns
df[columns_to_normalize] = scaler.fit_transform(df[columns_to_normalize])

# convert response to categorical
last_column = df.columns[-1]
df[last_column] = df[last_column].astype('category')
df.info()

from sklearn.preprocessing import LabelEncoder

# divide the data and encode the response
X = df[df.columns[:num_columns-1]]
y = df[df.columns[-1]]
label_encoder = LabelEncoder()
label_encoder.fit(['bad', 'good'])
```

```
y_encoded = label_encoder.transform(y)

####################################
# Scatter plots
####################################

import matplotlib.pyplot as plt
import seaborn as sns

axes = pd.plotting.scatter_matrix(X, alpha=0.5, figsize=(10, 10))
plt.tight_layout()
plt.savefig('scatter_matrix.png', dpi=300)

####################################
# show correlation heatmaps
####################################

dataplot = sns.heatmap(X.corr(), cmap="YlGnBu", annot=True)
fig = dataplot.get_figure()
fig.savefig("correlations.eps", format="eps", bbox_inches="tight")

# displaying heatmap
plt.show()


#################################################################
# Model 1: Logistic Regression
#################################################################

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, RepeatedKFold

# create model
model = LogisticRegression(max_iter=1000, random_state=0, solver='saga')

# define the hyperparameter grid
param_grid = {'C': [0.01, 0.05, 0.1, 0.5, 1, 5, 10],
              'penalty': ['l1', 'l2', 'elasticnet', 'none'],
              'l1_ratio': [0.1, 0.3, 0.5, 0.7, 0.9]}

# create cv object
cv = RepeatedKFold(n_splits=10, n_repeats=10, random_state=0)

# create grid-search cv object
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=cv, n_jobs=-1,
    scoring='accuracy')

# perform grid-search
grid_search.fit(X, y_encoded)

# get the best hyperparameters
best_params = grid_search.best_params_
best_score = grid_search.best_score_
best_model = grid_search.best_estimator_

# best hyperparameters
best_params

# best CV accuracy
best_score

from sklearn.metrics import accuracy_score

y_pred = best_model.predict(X)
# best model training error
accuracy_score(y_encoded, y_pred)

# get the coefficients of the trained model
```

```
coefficients = best_model.coef_[0]

# get the feature names
feature_names = X.columns

# create a dictionary to store the coefficients
feature_coefs = dict(zip(feature_names, coefficients))

# sort the feature coefficients by their absolute values in descending order
sorted_feature_importances = sorted(feature_coefs.items(), key=lambda x: abs(x[1]),
    reverse=True)

# print the feature coefficients
print("Feature Coefficients:")
for feature, coefficient in sorted_feature_importances:
    print(f"{feature}: Coefficient = {coefficient:.4f}")

# permuted VIM
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt

# initialize variables to store the permutation importances and standard deviations
all_importances = []
all_std_devs = []

# perform cross-validation
for train_index, test_index in cv.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y_encoded[train_index], y_encoded[test_index]

    # Fit the model on the training data
    model.fit(X_train, y_train)

    # Compute permuted variable importance measure for the test set
    result = permutation_importance(model, X_test, y_test, scoring='accuracy',
    n_repeats=10, random_state=0, n_jobs=-1)

    # Store the importances and standard deviations for each fold
    all_importances.append(result.importances_mean)
    all_std_devs.append(result.importances_std)

# compute the average importances and standard deviations across all folds
avg_importances = np.mean(all_importances, axis=0)
avg_std_devs = np.mean(all_std_devs, axis=0)

# create a DataFrame to store the VIM and standard deviations
feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': avg_importances,
    'Standard Deviation': avg_std_devs
})

# sort the feature importances in descending order
feature_importance = feature_importance.sort_values('Importance', ascending=True)

# plot the feature importances with standard deviations
ax = feature_importance.plot(x='Feature', y='Importance', kind='barh', figsize=(10,
    6), xerr='Standard Deviation', capsize=4)
ax.set_xlabel('Permutation Importance')
ax.set_title('Permutation Importance with Standard Deviation (Cross-Validation)')
plt.tight_layout()
plt.show()

# construct ALE plots
from alibi.explainers import ALE, plot_ale
logit_fun_lr = best_model.decision_function
proba_fun_lr = best_model.predict_proba
logit_ale_lr = ALE(logit_fun_lr, feature_names=feature_names, target_names=[str(y.
```

```
    name)])
proba_ale_lr = ALE(proba_fun_lr, feature_names=feature_names, target_names=[str(y.
    name)])
logit_exp_lr = logit_ale_lr.explain(X.values)
proba_exp_lr = proba_ale_lr.explain(X.values)
plot_ale(logit_exp_lr, n_cols=3, fig_kw={'figwidth': 8, 'figheight': 6}, sharey=None)

############################################################
# Model 2: Naive Bayes
############################################################

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV, RepeatedKFold

# create model
model = GaussianNB()

# define the hyperparameter grid
param_grid = {'var_smoothing':[1e-2, 5e-2, 1e-1, 5e-1, 1, 5, 10]}

# create cv object
cv = RepeatedKFold(n_splits=10, n_repeats=10, random_state=0)

# create grid-search cv object
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=cv, scoring='
    accuracy')

# perform grid-search
grid_search.fit(X, y_encoded)

# get the best hyperparameters
best_params = grid_search.best_params_
best_score = grid_search.best_score_
best_model = grid_search.best_estimator_

# best hyperparameters
best_params

# best CV accuracy
best_score

from sklearn.metrics import accuracy_score

y_pred = best_model.predict(X)
# best model training error
accuracy_score(y_encoded, y_pred)

# permuted VIM
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt

# initialize variables to store the permutation importances and standard deviations
all_importances = []
all_std_devs = []

# perform cross-validation
for train_index, test_index in cv.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y_encoded[train_index], y_encoded[test_index]

    # Fit the model on the training data
    model.fit(X_train, y_train)

    # Compute permuted variable importance measure for the test set
    result = permutation_importance(model, X_test, y_test, scoring='accuracy',
    n_repeats=10, random_state=0, n_jobs=-1)

    # Store the importances and standard deviations for each fold
```

```
    all_importances.append(result.importances_mean)
    all_std_devs.append(result.importances_std)

# compute the average importances and standard deviations across all folds
avg_importances = np.mean(all_importances, axis=0)
avg_std_devs = np.mean(all_std_devs, axis=0)

# create a DataFrame to store the VIM and standard deviations
feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': avg_importances,
    'Standard Deviation': avg_std_devs
})

# sort the feature importances in descending order
feature_importance = feature_importance.sort_values('Importance', ascending=True)

# plot the feature importances with standard deviations
ax = feature_importance.plot(x='Feature', y='Importance', kind='barh', figsize=(10,
    6), xerr='Standard Deviation', capsize=4)
ax.set_xlabel('Permutation Importance')
ax.set_title('Permutation Importance with Standard Deviation (Cross-Validation)')
plt.tight_layout()
plt.show()

#################################################################
# Model 3: LDA
#################################################################

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import cross_val_score, RepeatedKFold

# create model
model = LinearDiscriminantAnalysis(shrinkage='auto', solver='lsqr')

# create cv object
cv = RepeatedKFold(n_splits=10, n_repeats=10, random_state=0)

# get the cv score
cv_score = cross_val_score(model, X, y_encoded, cv=cv, scoring='accuracy')

# best CV accuracy
cv_score.mean()

# permuted VIM
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt

# initialize variables to store the permutation importances and standard deviations
all_importances = []
all_std_devs = []

# perform cross-validation
for train_index, test_index in cv.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y_encoded[train_index], y_encoded[test_index]

    # Fit the model on the training data
    model.fit(X_train, y_train)

    # Compute permuted variable importance measure for the test set
    result = permutation_importance(model, X_test, y_test, scoring='accuracy',
    n_repeats=10, random_state=0, n_jobs=-1)

    # Store the importances and standard deviations for each fold
    all_importances.append(result.importances_mean)
    all_std_devs.append(result.importances_std)

```

```
# compute the average importances and standard deviations across all folds
avg_importances = np.mean(all_importances, axis=0)
avg_std_devs = np.mean(all_std_devs, axis=0)

# create a DataFrame to store the VIM and standard deviations
feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': avg_importances,
    'Standard Deviation': avg_std_devs
})

# sort the feature importances in descending order
feature_importance = feature_importance.sort_values('Importance', ascending=True)

# plot the feature importances with standard deviations
ax = feature_importance.plot(x='Feature', y='Importance', kind='barh', figsize=(10,
    6), xerr='Standard Deviation', capsize=4)
ax.set_xlabel('Permutation Importance')
ax.set_title('Permutation Importance with Standard Deviation (Cross-Validation)')
plt.tight_layout()
plt.show()

#############################################################
# Model 4: SVM
#############################################################

from sklearn.model_selection import GridSearchCV, RepeatedKFold
from sklearn.neighbors import KNeighborsClassifier

# set model
knn = KNeighborsClassifier()

# search parameters
param_grid = {'n_neighbors': range(1, 31)}

# grid search to find the best parameter with K-fold CV.
kfold = RepeatedKFold(n_splits=10, n_repeats=5, random_state=0)
grid_search = GridSearchCV(knn, param_grid, cv=kfold, scoring='accuracy', verbose=1)

# Fit the model
grid_search.fit(X, y_encoded)

# best model
best_knn_params = grid_search.best_params_
print("Best Parameters: ", best_knn_params)
print("Best Cross-Validated Score", grid_search.best_score_)

# plot parameter tuning curve
cv_results_df = pd.DataFrame(grid_search.cv_results_)
n_neighbors = cv_results_df['param_n_neighbors']
CV_acc = cv_results_df['mean_test_score']

plt.figure(figsize=(10, 6))
plt.plot(n_neighbors, CV_acc, marker='o')
plt.title('Cross-Validated Accuracy vs. n_neighbors')
plt.xlabel('n_neighbors')
plt.ylabel('Mean CV Accuracy')
plt.grid(True)
plt.savefig("knn-tuning.png", format="png", bbox_inches="tight", dpi=300)
plt.show()

####### VIM importance
from sklearn.inspection import permutation_importance

# Initialize variables to store the permutation importances and standard deviations
all_importances = []
all_std_devs = []

```

```
# Perform cross-validation
for train_index, test_index in kfold.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y_encoded[train_index], y_encoded[test_index]

    # Fit the model on the training data
    knn.fit(X_train, y_train)

    # Compute permuted variable importance measure for the test set
    result = permutation_importance(knn, X_test, y_test, scoring='accuracy',
    n_repeats=10, random_state=0, n_jobs=-1)

    # Store the importances and standard deviations for each fold
    all_importances.append(result.importances_mean)
    all_std_devs.append(result.importances_std)

# Compute the average importances and standard deviations across all folds
avg_importances = np.mean(all_importances, axis=0)
avg_std_devs = np.mean(all_std_devs, axis=0)

# create a DataFrame to store the VIM and standard deviations
feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': avg_importances,
    'Standard Deviation': avg_std_devs
})

# sort the feature importances in descending order
feature_importance = feature_importance.sort_values('Importance', ascending=True)

# plot the feature importances with standard deviations
ax = feature_importance.plot(x='Feature', y='Importance', kind='barh', figsize=(10,
    6), xerr='Standard Deviation', capsize=4)
ax.set_xlabel('Permutation Importance')
ax.set_title('Permutation Importance with Standard Deviation (Cross-Validation)')
plt.tight_layout()
plt.savefig("knn-vim.png", format="png", bbox_inches="tight", dpi=300)
plt.show()

#################################################################
# Model 5: KNN
#################################################################

from sklearn.model_selection import GridSearchCV, RepeatedKFold
from sklearn.svm import SVC

# set model
svc = SVC()

#################
# 1st search
#################

# search parameters
param_grid = {'C': [0.1, 1, 10, 50, 100, 1000], 'kernel': ['rbf']}

# grid search to find the best parameter with K-fold CV.
kfold = RepeatedKFold(n_splits=10, n_repeats=5, random_state=0)
grid_search = GridSearchCV(svc, param_grid, cv=kfold, scoring='accuracy', verbose=1)

# Fit the model
grid_search.fit(X, y_encoded)

# best model
best_svc_params = grid_search.best_params_
print("Best Parameters: ", best_svc_params)
print("Best Cross-Validated Score", grid_search.best_score_)

```

```
#################
# 2nd search
#################

# search parameters
param_grid = {'C': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100], 'kernel': ['rbf']}

# grid search to find the best parameter with K-fold CV.
kfold = RepeatedKFold(n_splits=10, n_repeats=5, random_state=0)
grid_search = GridSearchCV(svc, param_grid, cv=kfold, scoring='accuracy', verbose=1)

# Fit the model
grid_search.fit(X, y_encoded)

# best model
best_svc_params = grid_search.best_params_
print("Best Parameters: ", best_svc_params)
print("Best Cross-Validated Score", grid_search.best_score_)

#################
# 3rd search
#################

# search parameters
param_grid = {'C': range(30,51), 'kernel': ['rbf']}

# grid search to find the best parameter with K-fold CV.
kfold = RepeatedKFold(n_splits=10, n_repeats=5, random_state=0)
grid_search = GridSearchCV(svc, param_grid, cv=kfold, scoring='accuracy', verbose=1)

# Fit the model
grid_search.fit(X, y_encoded)

# best model
best_svc_params = grid_search.best_params_
print("Best Parameters: ", best_svc_params)
print("Best Cross-Validated Score", grid_search.best_score_)


####### VIM importance

svc = SVC(**best_svc_params)

# Initialize variables to store the permutation importances and standard deviations
all_importances = []
all_std_devs = []

# Perform cross-validation
for train_index, test_index in kfold.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y_encoded[train_index], y_encoded[test_index]

    # Fit the model on the training data
    svc.fit(X_train, y_train)

    # Compute permuted variable importance measure for the test set
    result = permutation_importance(svc, X_test, y_test, scoring='accuracy',
    n_repeats=10, random_state=0, n_jobs=-1)

    # Store the importances and standard deviations for each fold
    all_importances.append(result.importances_mean)
    all_std_devs.append(result.importances_std)

# Compute the average importances and standard deviations across all folds
avg_importances = np.mean(all_importances, axis=0)
avg_std_devs = np.mean(all_std_devs, axis=0)

# create a DataFrame to store the VIM and standard deviations
```

```
feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': avg_importances,
    'Standard Deviation': avg_std_devs
})

# sort the feature importances in descending order
feature_importance = feature_importance.sort_values('Importance', ascending=True)

# plot the feature importances with standard deviations
ax = feature_importance.plot(x='Feature', y='Importance', kind='barh', figsize=(10, 6), xerr='Standard Deviation', capsize=4)
ax.set_xlabel('Permutation Importance')
ax.set_title('Permutation Importance with Standard Deviation (Cross-Validation)')
plt.tight_layout()
plt.savefig("svc-vim.png", format="png", bbox_inches="tight", dpi=300)
plt.show()

#################################################################
# Model 6: neural network
#################################################################

from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV, RepeatedKFold

# create model
model = MLPClassifier(random_state=0, activation='logistic', solver='adam', max_iter=1000, learning_rate='adaptive')

# define the hyperparameter grid
param_grid = {'hidden_layer_sizes':[(50,), (100,)],
              'alpha':[0.0001, 0.001, 0.01],
              'learning_rate_init':[0.01, 0.1]}

# create cv object
cv = RepeatedKFold(n_splits=10, n_repeats=1, random_state=0)

# create grid-search cv object
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=cv, scoring='accuracy')

# perform grid-search
grid_search.fit(X, y_encoded)

# get the best hyperparameters
best_params = grid_search.best_params_
best_score = grid_search.best_score_
best_model = grid_search.best_estimator_

# best hyperparameters
best_params

# best CV accuracy
best_score

from sklearn.metrics import accuracy_score

y_pred = best_model.predict(X)
# best model training error
accuracy_score(y_encoded, y_pred)

# permuted VIM
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt

# initialize variables to store the permutation importances and standard deviations
all_importances = []
all_std_devs = []
```

```
569
570 # perform cross-validation
571 for train_index, test_index in cv.split(X):
572     X_train, X_test = X.iloc[train_index], X.iloc[test_index]
573     y_train, y_test = y_encoded[train_index], y_encoded[test_index]
574
575     # Fit the model on the training data
576     model.fit(X_train, y_train)
577
578     # Compute permuted variable importance measure for the test set
579     result = permutation_importance(model, X_test, y_test, scoring='accuracy',
        n_repeats=10, random_state=0, n_jobs=-1)
580
581     # Store the importances and standard deviations for each fold
582     all_importances.append(result.importances_mean)
583     all_std_devs.append(result.importances_std)
584
585 # compute the average importances and standard deviations across all folds
586 avg_importances = np.mean(all_importances, axis=0)
587 avg_std_devs = np.mean(all_std_devs, axis=0)
588
589 # create a DataFrame to store the VIM and standard deviations
590 feature_importance = pd.DataFrame({
591     'Feature': X.columns,
592     'Importance': avg_importances,
593     'Standard Deviation': avg_std_devs
594 })
595
596 # sort the feature importances in descending order
597 feature_importance = feature_importance.sort_values('Importance', ascending=True)
598
599 # plot the feature importances with standard deviations
600 ax = feature_importance.plot(x='Feature', y='Importance', kind='barh', figsize=(10,
        6), xerr='Standard Deviation', capsize=4)
601 ax.set_xlabel('Permutation Importance')
602 ax.set_title('Permutation Importance with Standard Deviation (Cross-Validation)')
603 plt.tight_layout()
604 plt.show()
605
606 ################################################################
607 # Model 7: Classification Tree
608 ################################################################
609
610 from sklearn.tree import DecisionTreeClassifier
611 from sklearn.model_selection import GridSearchCV, RepeatedKFold
612
613 # set model
614 clt = DecisionTreeClassifier()
615
616 #################
617 # 1st search
618 #################
619
620 # search parameters
621 param_grid = {'max_depth': [5,10,15,20], 'min_samples_split': [2,3,5,10], '
        min_samples_leaf': [2,3,5,10], 'ccp_alpha': [0.0,0.001,0.01,0.1]}
622 kfold = RepeatedKFold(n_splits=10, n_repeats=5, random_state=0)
623 grid_search = GridSearchCV(clt, param_grid, cv=kfold, scoring='accuracy', verbose=1)
624 grid_search.fit(X, y)
625
626 # best model
627 best_clt_params = grid_search.best_params_
628 print("Best Parameters: ", best_clt_params)
629 print("Best Cross-Validated Score", grid_search.best_score_)
630
631 #################
632 # 2nd search
633 #################
```

```
634
635 # search parameters
636 param_grid = {'max_depth': [13,15,17,19], 'min_samples_split': [4,5,7,9], '
        min_samples_leaf': [2,3,4], 'ccp_alpha': [0.0,0.0001,0.0005,0.001]}
637 kfold = RepeatedKFold(n_splits=10, n_repeats=5, random_state=0)
638 grid_search = GridSearchCV(clt, param_grid, cv=kfold, scoring='accuracy', verbose=1)
639 grid_search.fit(X, y)
640
641 # best model
642 best_clt_params = grid_search.best_params_
643 print("Best Parameters: ", best_clt_params)
644 print("Best Cross-Validated Score", grid_search.best_score_)
645
646 #################
647 # 3rd search
648 #################
649
650 # search parameters
651 param_grid = {'max_depth': [14,15,16], 'min_samples_split': [6,7,8], '
        min_samples_leaf': [3,4], 'ccp_alpha': [0.0002,0.0003,0.0005,0.0007,0.0009]}
652 kfold = RepeatedKFold(n_splits=10, n_repeats=5, random_state=0)
653 grid_search = GridSearchCV(clt, param_grid, cv=kfold, scoring='accuracy', verbose=1)
654 grid_search.fit(X, y)
655
656 # best model
657 best_clt_params = grid_search.best_params_
658 print("Best Parameters: ", best_clt_params)
659 print("Best Cross-Validated Score", grid_search.best_score_)
660
661 #################
662 # 4th search
663 #################
664
665 # search parameters
666 param_grid = {'max_depth': [15,16,17], 'min_samples_split': [7,8,9], '
        min_samples_leaf': [3,4,5], 'ccp_alpha': [0.0006,0.0007,0.0008]}
667 kfold = RepeatedKFold(n_splits=10, n_repeats=5, random_state=0)
668 grid_search = GridSearchCV(clt, param_grid, cv=kfold, scoring='accuracy', verbose=1)
669 grid_search.fit(X, y)
670
671 # best model
672 best_clt_params = grid_search.best_params_
673 print("Best Parameters: ", best_clt_params)
674 print("Best Cross-Validated Score", grid_search.best_score_)
675
676
677 ######## Built-in feature_importance_ function
678 clt = DecisionTreeClassifier(**best_clt_params)
679 clt.fit(X,y)
680
681 feature_importances = clt.feature_importances_
682 feature_names = X.columns
683
684 # print feature_importance_
685 feature_importance_dict = dict(zip(feature_names, feature_importances))
686 ranked_features = sorted(feature_importance_dict.items(), key=lambda x: x[1], reverse
        =True)
687 print("Ranked Features by Importance:")
688 for rank, (feature, importance) in enumerate(ranked_features, start=1):
689     print(f"{rank}. {feature}: {importance:.4f}")
690
691
692 ####### VIM importance
693 clt = DecisionTreeClassifier(**best_clt_params)
694
695 # Initialize variables to store the permutation importances and standard deviations
696 all_importances = []
697 all_std_devs = []
```

```
698
699 # Perform cross-validation
700 for train_index, test_index in kfold.split(X):
701     X_train, X_test = X.iloc[train_index], X.iloc[test_index]
702     y_train, y_test = y_encoded[train_index], y_encoded[test_index]
703
704     # Fit the model on the training data
705     clt.fit(X_train, y_train)
706
707     # Compute permuted variable importance measure for the test set
708     result = permutation_importance(clt, X_test, y_test, scoring='accuracy',
        n_repeats=10, random_state=0, n_jobs=-1)
709
710     # Store the importances and standard deviations for each fold
711     all_importances.append(result.importances_mean)
712     all_std_devs.append(result.importances_std)
713
714 # Compute the average importances and standard deviations across all folds
715 avg_importances = np.mean(all_importances, axis=0)
716 avg_std_devs = np.mean(all_std_devs, axis=0)
717
718 # create a DataFrame to store the VIM and standard deviations
719 feature_importance = pd.DataFrame({
720     'Feature': X.columns,
721     'Importance': avg_importances,
722     'Standard Deviation': avg_std_devs
723 })
724
725 # sort the feature importances in descending order
726 feature_importance = feature_importance.sort_values('Importance', ascending=True)
727
728 # plot the feature importances with standard deviations
729 ax = feature_importance.plot(x='Feature', y='Importance', kind='barh', figsize=(10,
        6), xerr='Standard Deviation', capsize=4)
730 ax.set_xlabel('Permutation Importance')
731 ax.set_title('Permutation Importance with Standard Deviation (Cross-Validation)')
732 plt.tight_layout()
733 plt.savefig("ct-vim.png", format="png", bbox_inches="tight", dpi=300)
734 plt.show()
735
736 ################################################################
737 # Model 8: Light GBM
738 ################################################################
739
740 import lightgbm as lgb
741 from sklearn.model_selection import GridSearchCV, RepeatedKFold
742 import warnings
743
744 # set model
745 lgbc = lgb.LGBMClassifier(num_threads=8, verbose=-1)
746
747 # warnings.filterwarnings("ignore", category=UserWarning, module="lightgbm")
748
749 #################
750 # 1st search
751 #################
752
753 param_grid = {
754     'learning_rate': [0.01, 0.1, 0.5],
755     'n_estimators': [200, 300],
756     'max_depth': [10, 20, 30],
757     'num_leaves': [15, 31, 63],
758 }
759
760
761 kfold = RepeatedKFold(n_splits=10, n_repeats=5, random_state=0)
762 grid_search = GridSearchCV(lgbc, param_grid, cv=kfold, scoring='accuracy', verbose=1)
763 grid_search.fit(X, y)
```

```
764
765 # best model
766 best_lgbc_params = grid_search.best_params_
767 print("Best Parameters: ", best_lgbc_params)
768 print("Best Cross-Validated Score", grid_search.best_score_)
769
770 #################
771 # 2nd search
772 #################
773
774 param_grid = {
775     'learning_rate': [0.05, 0.1, 0.2],
776     'n_estimators': [300, 400],
777     'max_depth': [30, 40],
778     'num_leaves': [63, 96],
779 }
780
781
782 kfold = RepeatedKFold(n_splits=10, n_repeats=5, random_state=0)
783 grid_search = GridSearchCV(lgbc, param_grid, cv=kfold, scoring='accuracy', verbose=1)
784 grid_search.fit(X, y)
785
786 # best model
787 best_lgbc_params = grid_search.best_params_
788 print("Best Parameters: ", best_lgbc_params)
789 print("Best Cross-Validated Score", grid_search.best_score_)
790
791 #################
792 # 3rd search
793 #################
794
795 param_grid = {
796     'learning_rate': [0.18, 0.2, 0.22],
797     'n_estimators': [300, 400],
798     'max_depth': [30],
799     'num_leaves': [96],
800 }
801
802
803 kfold = RepeatedKFold(n_splits=10, n_repeats=5, random_state=0)
804 grid_search = GridSearchCV(lgbc, param_grid, cv=kfold, scoring='accuracy', verbose=1)
805 grid_search.fit(X, y)
806
807 # best model
808 best_lgbc_params = grid_search.best_params_
809 print("Best Parameters: ", best_lgbc_params)
810 print("Best Cross-Validated Score", grid_search.best_score_)
811
812 #################
813 # 4th search
814 #################
815
816 param_grid = {
817     'learning_rate': [0.22, 0.24, 0.26, 0.28, 0.3],
818     'n_estimators': [300, 400, 500],
819     'max_depth': [30],
820     'num_leaves': [96],
821 }
822
823 kfold = RepeatedKFold(n_splits=10, n_repeats=5, random_state=0)
824 grid_search = GridSearchCV(lgbc, param_grid, cv=kfold, scoring='accuracy', verbose=1)
825 grid_search.fit(X, y)
826
827 # best model
828 best_lgbc_params = grid_search.best_params_
829 print("Best Parameters: ", best_lgbc_params)
830 print("Best Cross-Validated Score", grid_search.best_score_)
831
```

```
832
833 ####### built-in feature importance
834 lgbc = lgb.LGBMClassifier(**best_lgbc_params, num_threads=8, verbose=-1)
835 lgbc.fit(X,y)
836
837 lgb.plot_importance(lgbc, importance_type="gain", figsize=(7,6), title="LGBM Feature
        Importance")
838 plt.savefig("lgbm-importance.png", format="png", bbox_inches="tight", dpi=300)
839 plt.show()
840
841
842 ####### VIM importance
843 lgbc = lgb.LGBMClassifier(**best_lgbc_params, num_threads=8, verbose=-1)
844
845 # Initialize variables to store the permutation importances and standard deviations
846 all_importances = []
847 all_std_devs = []
848
849 # Perform cross-validation
850 for train_index, test_index in kfold.split(X):
851     X_train, X_test = X.iloc[train_index], X.iloc[test_index]
852     y_train, y_test = y_encoded[train_index], y_encoded[test_index]
853
854     # Fit the model on the training data
855     lgbc.fit(X_train, y_train)
856
857     # Compute permuted variable importance measure for the test set
858     result = permutation_importance(lgbc, X_test, y_test, scoring='accuracy',
        n_repeats=10, random_state=0, n_jobs=-1)
859
860     # Store the importances and standard deviations for each fold
861     all_importances.append(result.importances_mean)
862     all_std_devs.append(result.importances_std)
863
864 # Compute the average importances and standard deviations across all folds
865 avg_importances = np.mean(all_importances, axis=0)
866 avg_std_devs = np.mean(all_std_devs, axis=0)
867
868 # create a DataFrame to store the VIM and standard deviations
869 feature_importance = pd.DataFrame({
870     'Feature': X.columns,
871     'Importance': avg_importances,
872     'Standard Deviation': avg_std_devs
873 })
874
875 # sort the feature importances in descending order
876 feature_importance = feature_importance.sort_values('Importance', ascending=True)
877
878 # plot the feature importances with standard deviations
879 ax = feature_importance.plot(x='Feature', y='Importance', kind='barh', figsize=(10,
        6), xerr='Standard Deviation', capsize=4)
880 ax.set_xlabel('Permutation Importance')
881 ax.set_title('Permutation Importance with Standard Deviation (Cross-Validation)')
882 plt.tight_layout()
883 plt.savefig("lgbm-vim.png", format="png", bbox_inches="tight", dpi=300)
884 plt.show()
885
886 ################################################################
887 # Model 9: Random Forest
888 ################################################################
889
890 from sklearn.ensemble import RandomForestClassifier
891 from sklearn.model_selection import GridSearchCV, RepeatedKFold
892
893 # create model
894 model = RandomForestClassifier(random_state=0, n_jobs=-1)
895
896 # define the hyperparameter grid
```

```
param_grid = {'n_estimators':[250, 300, 350],
              'max_depth':[20, 25, 30],
              'min_samples_split':[2, 5, 10],
              'min_samples_leaf':[1, 2, 5]}

# create cv object
cv = RepeatedKFold(n_splits=10, n_repeats=1, random_state=0)

# create grid-search cv object
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=cv, scoring='
    accuracy')

# perform grid-search
grid_search.fit(X, y_encoded)

# get the best hyperparameters
best_params = grid_search.best_params_
best_score = grid_search.best_score_
best_model = grid_search.best_estimator_

# best hyperparameters
best_params

# best CV accuracy
best_score

from sklearn.metrics import accuracy_score

y_pred = best_model.predict(X)
# best model training error
accuracy_score(y_encoded, y_pred)

# get the feature importances from the best model
importances = best_model.feature_importances_

# sort the feature importances in decending order
indices = np.argsort(importances)[::-1]

# rearrange the feature names based on the sorted importances
feature_names_new = [X.columns[i] for i in indices]

# create a bar plot
plt.figure(figsize=(10, 6))
plt.title("Feature Importances")
plt.barh(range(len(importances)), importances[indices])
plt.yticks(range(len(importances)), feature_names_new)
plt.xlabel("Importance")
plt.ylabel("Features")
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()

# initialize variables to store the permutation importances and standard deviations
all_importances = []
all_std_devs = []

# perform cross-validation
for train_index, test_index in cv.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y_encoded[train_index], y_encoded[test_index]

    # Fit the model on the training data
    model.fit(X_train, y_train)

    # Compute permuted variable importance measure for the test set
    result = permutation_importance(model, X_test, y_test, scoring='accuracy',
    n_repeats=10, random_state=0, n_jobs=-1)

```

```
    # Store the importances and standard deviations for each fold
    all_importances.append(result.importances_mean)
    all_std_devs.append(result.importances_std)

# compute the average importances and standard deviations across all folds
avg_importances = np.mean(all_importances, axis=0)
avg_std_devs = np.mean(all_std_devs, axis=0)

# create a DataFrame to store the VIM and standard deviations
feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': avg_importances,
    'Standard Deviation': avg_std_devs
})

# sort the feature importances in descending order
feature_importance = feature_importance.sort_values('Importance', ascending=True)

# plot the feature importances with standard deviations
ax = feature_importance.plot(x='Feature', y='Importance', kind='barh', figsize=(10,
    6), xerr='Standard Deviation', capsize=4)
ax.set_xlabel('Permutation Importance')
ax.set_title('Permutation Importance with Standard Deviation (Cross-Validation)')
plt.tight_layout()
plt.show()
```