Machine Learning Model Development for Classification of Raisin Grains

Technical Report

Developed by Nayanathara Widyalankara (Reg. No- 257) for the completion of the capstone project of the Machine learning foundation Course- Batch 2

2022.04.30

Table of Contents

1 Introduction

In the project, a machine learning model was developed to predict and classify the Raisin grains in to two different varieties Kecimen and Besni which are grown in Turkey. Firstly, a total of 900 pieces raisin grains were obtained, from an equal number of both varieties. These images were subjected to various preprocessing steps and 7 morphological feature extraction operations were performed using image processing techniques. The distributions of both raisin varieties on the features were examined and these distributions were also studied. Later, models were created using different classification models and selected the model which has the highest accuracy. Considering the results obtained, it is possible to say that the project was successful.

2 Dataset

Dataset name: Raisin Dataset Data Set

Repository: UCI machine learning repository

Data Set Characteristics:	Multivariate	**Number of Instances:**	900	**Area:**	Life
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	8	**Date Donated**	2021-04-01
Associated Tasks:	Classification	**Missing Values?**	N/A	**Number of Web Hits:**	1300561

Data Set Information: Images of Kecimen and Besni raisin varieties grown in Turkey were obtained with CVS. A total of 900 raisin grains were used, including 450 pieces from both varieties. These images were subjected to various stages of pre-processing and 7 morphological features were extracted.

2.1 Attribute Information

1.) **Area:** Gives the number of pixels within the boundaries of the raisin
2.) **Perimeter:** It measures the environment by calculating the distance between the boundaries of the raisin and the pixels around it
3.) **MajorAxisLength:** Gives the length of the main axis, which is the longest line that can be drawn on the raisin
4.) **MinorAxisLength:** Gives the length of the small axis, which is the shortest line that can be drawn on the raisin
5.) **Eccentricity:** It gives a measure of the eccentricity of the ellipse, which has the same moments as raisins
6.) **ConvexArea:** Gives the number of pixels of the smallest convex shell of the region formed by the raisin.
7.) **Extent:** Gives the ratio of the region formed by the raisin to the total pixels in the bounding box
8.) **Class:** Kecimen and Besni raisin

3 Methodology

3.1 Platform

The 'Googles Colab' python development environment was used for coding this Machine Learning project.

3.2 Libraries

- Pandas – This library was used for loading, analyzing and manipulation of data.
- NumPy – Used to perform mathematical operations (on arrays and matrices).
- Matplotlib – Extension of NumPy used for plotting
- Seaborn – Used for data visualization
- Scikit-learn - Library used for machine learning. Includes ML models and other necessary functions and tools.

3.3 Creating the Dataframe

Dataset was loaded into the Colab environment from github repository using the Pands `read_csv` feature.

`data.columns` was used to get the column names of the dataframe.

```
data.columns
```

```
Index(['Area', 'MajorAxisLength', 'MinorAxisLength', 'Eccentricity',
       'ConvexArea', 'Extent', 'Perimeter', 'Class'],
      dtype='object')
```

3.4 Data Exploration

- Exploration started by checking the dimensions of the dataframe. The result was that it contained 8 columns (7 morphological features and class of the raisin variety) and 900 rows. This matched with the dataset description confirmation that all data has been imported.
- `data['Class'].value_counts()` has again validated the number of observations for raisin varieties as per mentioned in the description. Therefore, it can be said that data set in balanced
- Next the data types were inspected. And the Summery of the dataframe was taken using `describe`. It will give an overall picture of the statistical information of data features.

```
Area               int64
MajorAxisLength  float64
MinorAxisLength  float64
Eccentricity     float64
ConvexArea         int64
Extent           float64
Perimeter        float64
Class             object
dtype: object
```

- Then the dataset was checked for any missing values with `data.isnull().any()` and as per the result there were none and can be versified the result with `info()`.
- `data.duplicated()function` is used to identify duplicates if any.

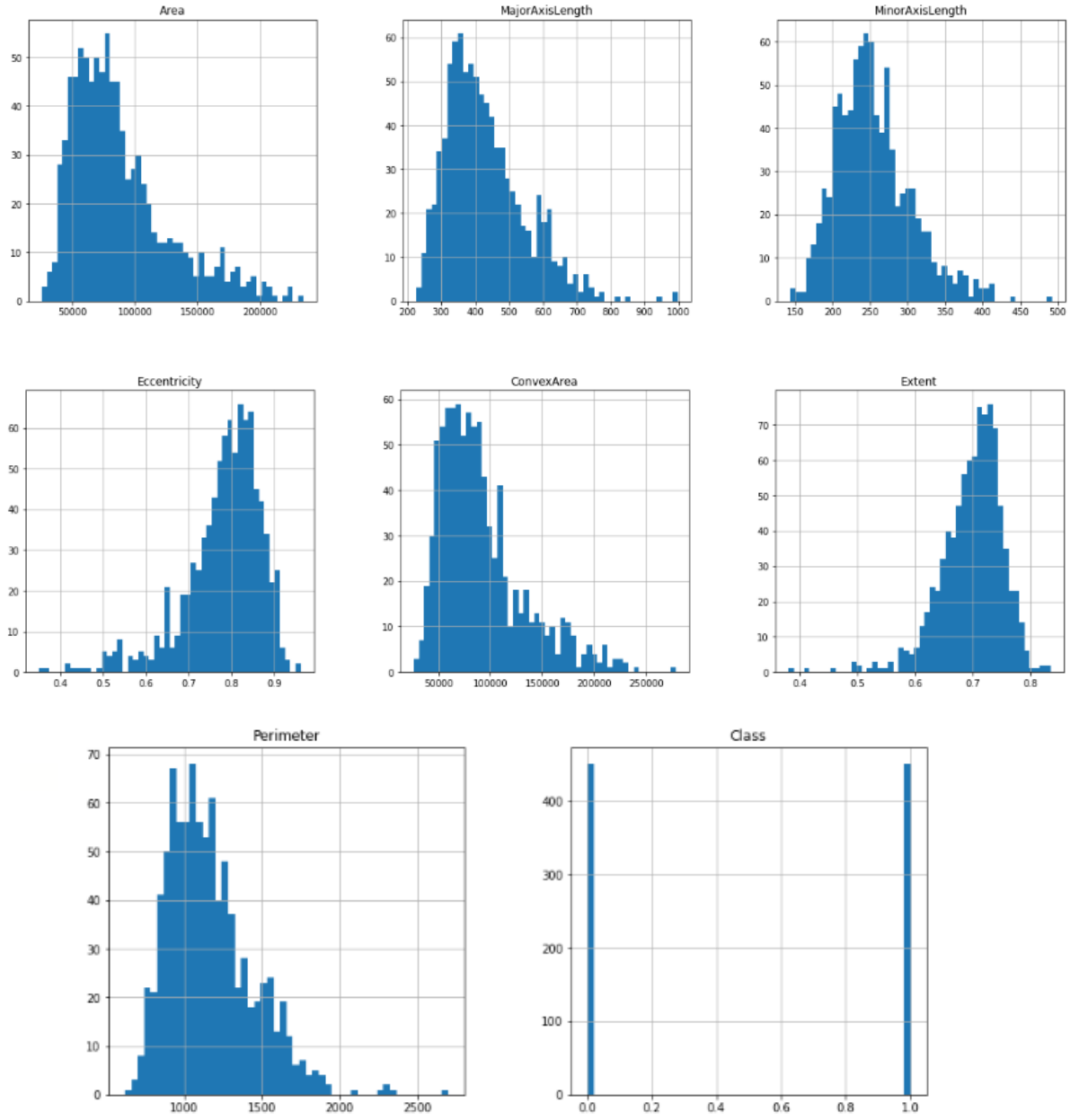
3.5 Converting Categorical values into numerical values

The outcome column – `Class` was in type 'Object' [categorical]. It was converted into a numerical variable using below function.

```
data['Class'] = data['Class'].replace({"Kecimen": 0, "Besni": 1})
```

3.6 Study the Data Behavior

Histograms were used to observed how the values have spread within each attribute. `data.hist(bins=50, figsize=(20, 20))` was used for plotting the graphs.



`skew()` is used to check the Skewness of the data features and It was observed that almost all the feature distributions are skewed as the skewness values are greater than 1 or less than -1 indicates a highly skewed distribution.

```
Area              1.175237
MajorAxisLength   0.989544
MinorAxisLength   0.800049
Eccentricity     -1.327503
ConvexArea        1.242904
Extent           -1.151505
Perimeter         1.017761
Class             0.000000
dtype: float64
```

3.7 Treat Outliers

- To check for outliers, box plots were used. From the results we can observe that all the features have outliers except for `Class` since it is a derived numerical value obtained from categorical variable .
- Therefore, the next step of methodology was to remove the outliers. For that Inter Quartile Range approach was used to find the outliers since it is the most commonly used and most trusted approach used in the research field.

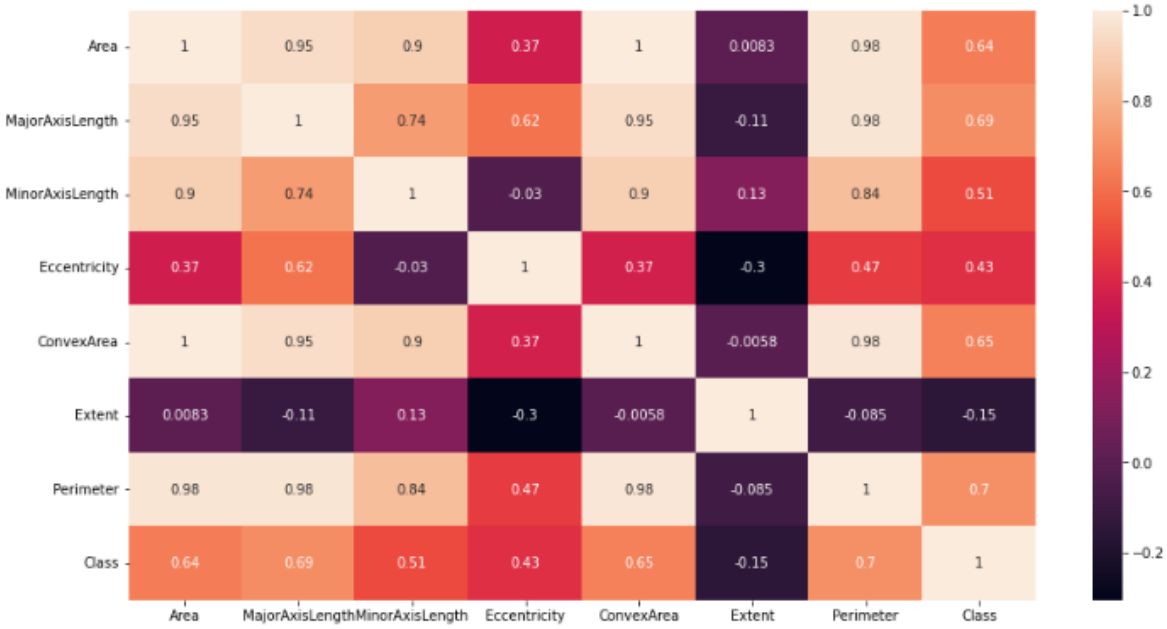
```
outliers=['Area','MajorAxisLength','MinorAxisLength', 'Eccentricity','ConvexArea','Extent', 'Perimeter']
print("Old Shape: ", data.shape)
for feature in outliers:
  Q1= np.percentile(data[feature], 25,
                    interpolation = 'midpoint')
  Q3 = np.percentile(data[feature], 75,
                    interpolation = 'midpoint')
  IQR = Q3- Q1
  upper= Q3+1.5*IQR
  lower= Q1-1.5*IQR
  data=data[(data[feature]>lower) & (data[feature]<upper)]

print("New Shape: ", data.shape)
```

```
Old Shape:  (900, 8)
New Shape:  (778, 8)
```

- With the results obtained 122 data points were removed from the dataset and checked again the number of observations for raisin varieties to make sure that the data set is in balanced. With the results obtained varieties are in 1.1:1 ratio and can be validated that they are in acceptable level.

3.8 Feature Selection

- `'Class'` was selected as the Y variable since the model will be predicting it.
- To select the X variables following steps were followed.
- Correlation matrix was generated using `correlation_matrix = data.corr()` and the heatmap of correlation matrix was generated using `sns.heatmap(correlation_matrix, annot=True)`



- Relationship between the features were inspected again using pair plot `sns.pairplot(data)`
- `sns.pairplot(data, hue='Class', markers='+')` was used to map the data distribution with the `'Class'` output.

Following summarized results were used to select features.

Correlation Coefficient

index	Class
Area	0.6435420445134135
MajorAxisLength	0.6892760521343054
MinorAxisLength	0.5068807880069588
Eccentricity	0.4336764604191128
ConvexArea	0.6542336830091916
Extent	-0.14683495342460995
Perimeter	0.6956906572398455
Class	1.0

The columns which have a correlation above 0.5 with the Y variable can be considered as good features. So that we can drop `'Eccentricity'` and `'Extent'` from X variables.

Interclass Correlation Coefficient

Following feature pairs have high Interclass Correlation Coefficient (>=0.98) which can lead only to select one feature from the pair.

- ConvexArea and Perimeter
- ConvexArea and Area
- Area and Perimeter
- MajorAxisLength and Perimeter

So, the X variables can be selected as `MinorAxisLength` and `ConvexArea`

```
X_variables = ['MinorAxisLength', 'ConvexArea']
y_variable = Class
```

3.9 Model Development

3.9.1 Train Test Split

The dataset was slip into two. One for training the model and the other for testing the model. Training dataset includes 70% of the original set. Split was done using the scikit learn function `train_test_split` the outputs are `X_train, X_test, y_train, y_test`.

3.9.2 Model Building

A function was written for training the model. Inputs for the training function are the `model` (imported from Scikit Learn), `model_name`, `X_train`, `y_train`, `X_test` and `y_test`. `model.fit` was used to push the `X_train` and `y_train` data into the model and train it. `model.predict` was used to predict the `'Class'` for the `X_test` data.

Scikitlearn metrics scoring functions were using to score the model output. They are `'accuracy'`, `'precision'`, `'f1_score'`, and Area under the roc curve- `'roc_auc'`. The function returns the `'model_name'`, `'model'`, scoring functions, actual test dataset- `'y act'` and predicted data set- `'y pred'`.

```
def model_train(model, model_name, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    y_pred_prob = model.predict_proba(X_test)[:, 1]
    test_result = pd.DataFrame(data={'y_act':y_test, 'y_pred':y_pred, 'y_pred_prob':y_pred_prob})

    accuracy = metrics.accuracy_score(test_result['y_act'], test_result['y_pred'])
    precision = metrics.precision_score(test_result['y_act'], test_result['y_pred'], average='binary', pos_label=1)
    f1_score = metrics.f1_score(test_result['y_act'], test_result['y_pred'], average='weighted')  #weighted accounts
    roc_auc = metrics.roc_auc_score(test_result['y_act'], test_result['y_pred_prob'])



    return ({'model_name':model_name,
                'model':model,
                'accuracy':accuracy,
                'precision':precision,
               'f1_score':f1_score,
               'roc_auc':roc_auc,
               'y act': y_test,
               'y pred': y_pred
               })
```

3.9.3 Best model selection

Since this is a classification problem, several ML classification model were trained and scored to select the model with best predictions. This was done by creating a list called ‘models’. Each element

of the list the code calling the model training function for different ML models. Below is the list of models and the output from the model train function.

```
models = []
models.append(model_train(LogisticRegression(n_jobs=3, random_state = 0), 'lgr1', X_train, y_train, X_test, y_test))
models.append(model_train(KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2), 'KNeBR', X_train, y_train, X_test, y_test))
models.append(model_train(GaussianNB(), 'Gaus', X_train, y_train, X_test, y_test))
models.append(model_train(RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0), 'RF', X_train, y_train, X_test, y_test))
models.append(model_train(DecisionTreeClassifier(criterion = 'gini', random_state = 0 ), 'DeciT-G', X_train, y_train, X_test, y_test))
models.append(model_train(DecisionTreeClassifier(criterion = 'entropy', random_state = 0 ), 'DeciT-En', X_train, y_train, X_test, y_test))
models = pd.DataFrame(models)
models
```

	model_name	model	accuracy	precision	f1_score	roc_auc	y act	y pred
0	lgr1	LogisticRegression(n_jobs=3, random_state=0)	0.893162	0.863636	0.893349	0.946549	[1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, ...	[0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, ...
1	KNeBR	KNeighborsClassifier()	0.807692	0.767857	0.808112	0.869214	[1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, ...	[0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, ...
2	Gaus	GaussianNB()	0.833333	0.817308	0.833256	0.908379	[1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, ...	[0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, ...
3	RF	(DecisionTreeClassifier(criterion='entropy', m...	0.871795	0.850467	0.871899	0.918051	[1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, ...	[0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, ...
4	DeciT-G	DecisionTreeClassifier(random_state=0)	0.786325	0.720000	0.786637	0.792913	[1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, ...	[0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, ...
5	DeciT-En	DecisionTreeClassifier(criterion='entropy', ra...	0.807692	0.754237	0.808196	0.811406	[1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, ...	[0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, ...

Here we can see that the **logistic regression** has the most impressive scores. Which concludes that it is the best model for the development.

3.9.4 Tuning the Best model

Hyperparameters of the selected model were tuned using the grid search method.

```
# perparameters search for Logistic regression

parameters = {'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'], 'penalty': ['none', 'l1', 'l2', 'elasticnet']}
gs_model = GridSearchCV(LogisticRegression(), parameters)
gs_model.fit(X_train, y_train)
```

After selecting the best parameters, the Best model score was also obtained.

```
# Print the tuned parameters and score
print("Tuned Model Parameters: {}".format(gs_model.best_params_))
print("Best model score: {}".format(gs_model.best_score_))
```

```
Tuned Model Parameters: {'penalty': 'none', 'solver': 'lbfgs'}
Best model score: 0.8235643900781515
```

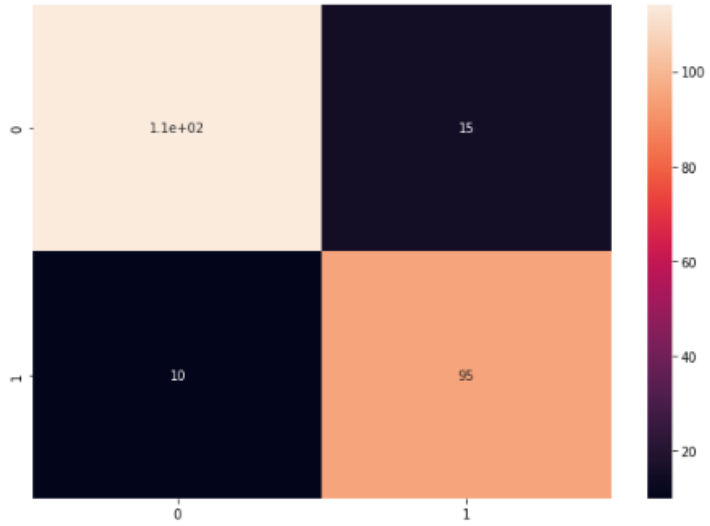
3.9.5 Scoring the model

Scoring functions were using to score the model output.

```
y_pred = gs_model.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.88      0.90       129
           1       0.86      0.90      0.88       105

    accuracy                           0.89       234
   macro avg       0.89      0.89      0.89       234
weighted avg       0.89      0.89      0.89       234

[[114  15]
 [ 10  95]]
```

Heat map of confusion matrix also obtained to graphically see the scores of the model.



Note: Hyper parameter tuning was done several other models as well. It validated that the **logistic regression** has the best recall and F1 scores.

3.9.6 Saving the model

After tuning the models, **Logistic Regression** was selected as the best model as it has the best recall and f1 scores. In order to store the trained model, pickle was used.

```
[159] import pickle

      save_file = 'model_lgr1.pickle'
      pickle.dump(model, open(save_file, 'wb'))
```

```
# loading from file
model_ = pickle.load(open(save_file, 'rb'))
model_
```

```
LogisticRegression(n_jobs=3, random_state=0)
```

4 Results

After following the methodology, the **Logistic Regression** model was selected as the best model and following are the results that have obtained from the scoring metrices.

```
              precision    recall  f1-score   support

           0       0.92      0.88      0.90       129
           1       0.86      0.90      0.88       105

    accuracy                           0.89       234
   macro avg       0.89      0.89      0.89       234
weighted avg       0.89      0.89      0.89       234

[[114  15]
 [ 10  95]]
```

Summary of the results:

Accuracy	0.89
Precision	0.89
Recall	0.89
F1-score	0.89
roc_auc	0.95 (Obtained from 3.9.3)

5 Conclusion

- The dataset was in balance with reasonable distribution of raisin varieties which results in building an unbiased model.
- Removing outliers of the dataset with “Inter Quartile Range approach” have validated the results of the model.
- Even though data were collected for several features, “MinorAxisLength” and “ConvexArea” were the features selected to build the model due to correlation of the variables.
- Among the different types of algorithms, “logistic regression” has the most impressive scores. Which concludes that, it is the best model for the development.
- Based on the results, in the confusion matrix which is used to visualize the accuracy of a classifier by comparing the actual and predicted classes, The accuracy of the model is the 89%. F1 score, which is the weighted average score of the true positive (recall) and precision is also 89%. These results concluded that the model can accurately predict and classify the Raisin grains in to two different varieties “Kecimen” and “Besni”.
- Model has 92% of true positive rate and 86% true negative rate. This concludes that the model can accurately predicts “Besni” variety than the “Kecimen” variety.
- With these results, it had achieved the objective the variety of classify the Raisin grains and further tuning can be done before deployment.

6 Discussion

When the results obtained from the study are evaluated, higher classification successes can be achieved by increasing the number of images or features obtained from the products and adding color, shape and texture features in addition to the morphological features.

In addition to the machine learning techniques used in the current study, studies can be carried out with other techniques or hybrid models.

The varieties of raisins used in the study is one of the products produced and exported in countries. Benefiting the image processing techniques, data sets can be created using the derived feature inferences with the machine learning models. In this way, automatic systems can be designed by using the created data sets for classification, calibration of the products or to be used in different processing stages.