CS5014 - Practical 2
170025298 11/04/2021

Introduction

This practical aims to solve a specific problem about identifying the texture and colour of an object of the image with a more detailed model. The practical investigates a more precise model by using different algorithms. Validation of the model includes testing the balanced accuracy of the model and investigating the confusion matrix for easier identification of the model's problem.

EXT. During testing, I found that the position of the boarder does affect the accuracy of the model, so I added x and y property to the training.

Design

Data Pre-processing

Pre-processing data is one of the most important stages of training the model. With data pre-processing, data that may lead to training inaccuracy or model segment fault will be removed, which keeps the model's stability. Data pre-processing stage does the following things:

- Clean missing data
- Clean error data
- Clean useless data
- Normalize data in the dataset
- Convert texture labels into numerical or binary (With one-hot encoding)

Common pre-processing

Common pre-processing dataset do the data cleaning and data scaling.

Since the dataset may contain useless data (For example, the image id and the object id), unexpected symbols (For example: want the column type is float, but get object actually), missing data (One blank is left Nan) or error values (For example, the width or height of the border is set 0, which is unable to recognize the object, but get labels for that), we may tackle with these data by fill correct data to the dataset, delete the row with missing data and correct the error data in the dataset.

These data value errors can be solved in the following steps:

- We can directly use the DROP function to drop the row or column of the training set with error values
- We can use the $fillNan()$ function to fill 0 to the empty area of the testing set (Whereas the set index should not be modified in case of uploading)

Scaling of the data decreases the hardness of scaling calculation and will decrement the effort of invalid data. Data are scaling scales the training and testing dataset separately. Scaling with training data will also record the mean value and expected value of the dataset and apply the scaling parameters to the testing set. Scaling of the test set uses the axiom as follows:

$$X_f = (X - u)/S$$

Whereas u is the mean value of the column and S is the standard value of the column. The result is shown as follows:

In [19]:
```
train_data
```

Out[19]:

	w	h	color	texture	lightness_0_0	lightness_0_1	lightness_0_2	lightness_1_0	lightness_1_1	lightness_1_2	...	bimp_8_16_double_2_2	bimp_8_
0	40	149	10	4	1.603943	1.642686	1.083023	1.221399	1.490157	0.770229	...	0.812073	
1	239	104	10	4	-0.365652	0.320397	0.313413	0.134012	0.250016	0.388352	...	-0.321708	
2	501	140	3	6	0.510014	0.359951	0.498426	0.470813	0.575922	0.648156	...	1.166238	
3	497	211	3	6	0.003405	0.012278	0.446403	-0.613373	-0.547187	0.285062	...	-0.507001	
4	92	271	10	4	0.540338	0.576459	0.423842	0.492813	0.693767	0.703751	...	-1.336727	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1329	236	460	10	9	0.596864	0.639873	0.570702	0.473347	0.618955	0.662411	...	0.508703	
1330	237	329	5	1	0.859428	-0.921894	-1.492874	1.076933	-0.665414	-1.225339	...	-0.150325	
1331	17	45	5	9	-0.107922	-1.502721	-1.035953	-0.357530	-1.247549	-1.261867	...	0.525152	
1332	34	84	3	9	-1.474453	-1.961828	-1.770515	-0.240884	-1.009302	-1.029288	...	-0.105795	
1333	95	94	4	1	1.006575	0.098948	-0.336558	-0.630228	-0.831231	0.456541	...	0.673594	

1334 rows × 445 columns

In [21]:
```
test_data
```

Out[21]:

	w	h	lightness_0_0	lightness_0_1	lightness_0_2	lightness_1_0	lightness_1_1	lightness_1_2	lightness_2_0	lightness_2_1	...	bimp_8_16_double_
0	137	112	-1.842668	1.065678	-0.550926	-1.125997	0.987674	1.592738	-1.117443	-0.427895	...	0.01
1	497	153	-2.647725	-2.648736	-2.416573	-2.190115	-2.413608	-1.849915	-1.158822	-0.915602	...	0.02
2	64	171	-0.558118	-0.997558	-0.051146	0.371513	-0.041996	-0.564416	-0.225211	-0.531917	...	-0.22
3	229	72	-0.602412	-0.596612	-0.394688	-0.561845	-0.640970	-0.453627	-0.540166	-0.694216	...	0.39
4	147	118	-0.371204	-0.685257	0.528858	-1.132490	-2.127821	0.020038	-0.323090	-1.574699	...	-1.14
...	...	...	...	...	...	...	...	...	...	...	...	
329	76	32	1.033392	1.428060	1.168150	1.182886	1.637316	1.388647	0.759466	1.316100	...	0.84
330	162	113	0.006776	-1.470211	-1.879195	-0.801818	-0.485536	-1.772178	-1.038211	-1.056922	...	-1.21
331	484	163	0.501205	-0.160883	-0.187002	1.682973	0.370376	-0.025674	1.778239	1.720513	...	-0.09
332	260	147	1.551354	-0.569227	0.416842	-0.534495	-1.940168	-1.280114	-0.023349	-0.461896	...	-0.85
333	498	331	0.149017	0.225299	0.337923	0.862474	0.032186	-0.081304	0.659827	-0.004500	...	-2.86

334 rows × 443 columns

Pre-processing General

```
In [8]: color_labels
Out[8]: ['beige',
         'black',
         'blue',
         'brown',
         'gray',
         'green',
         'orange',
         'pink',
         'purple',
         'red',
         'white',
         'yellow']
```

```
In [9]: texture_labels
Out[9]: ['barbed',
         'blurry',
         'coarse',
         'crusty',
         'fluffy',
         'fuzzy',
         'grassy',
         'gravel',
         'rippled',
         'smooth']
```

Since the dataset has labels represents showing above, we would convert those labels numerically. We use the *get_dummies*() function retrieving all labels of the dataset and converting labels into numeric according to the label's position in the list of labels. Hence we can get the result shows as following:

In [10]: train_data

Out[10]:

	w	h	color	texture	lightness_0_0	lightness_0_1	lightness_0_2	lightness_1_0	lightness_1_1	lightness_1_2	...	bimp_8_16_double_2_2	bimp_8_
0	40	149	10	4	217.989746	222.322266	187.733887	201.979736	221.929688	172.224609	...	0.050589	
1	239	104	10	4	119.215332	153.845947	148.174561	147.168945	153.015869	152.171143	...	-0.132715	
2	501	140	3	6	163.129639	155.894287	157.684570	164.145752	171.126221	165.814209	...	0.107848	
3	497	211	3	6	137.723389	137.889648	155.010498	109.496338	108.715820	146.747070	...	-0.162672	
4	92	271	10	4	164.650391	167.106445	153.850830	165.254639	177.674805	168.733643	...	-0.296817	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1330	236	460	10	9	167.485107	170.390381	161.399658	164.273438	173.517578	166.562744	...	0.001542	
1331	237	329	5	1	180.652588	89.512451	55.328125	194.697754	102.145996	67.431641	...	-0.105007	
1332	17	45	5	9	132.140381	59.433594	78.814697	122.392334	69.797119	65.513428	...	0.004201	
1333	34	84	3	9	63.609375	35.658203	41.056885	128.271973	83.036377	77.726807	...	-0.097807	
1334	95	94	4	1	188.031982	142.377930	114.764893	108.646729	92.931641	155.751953	...	0.028200	

1335 rows × 445 columns

Pre-processing with One-Hot encoding

For a linear model, if labels are converted into numeric, the training process may consider those numbers due to calculation from all other parts, which will lead to training confusion. As a result, we use one-hot encoding to encode texture and colour labels.

```
In [7]: train_data
```

Out[7]:

lightness_1_0	lightness_1_1	lightness_1_2	lightness_2_0	lightness_2_1	...	barbed	blurry	coarse	crusty	fluffy	fuzzy	grassy	gravel	rippled	smooth
201.979736	221.929688	172.224609	117.240723	195.561279	...	0	0	0	0	1	0	0	0	0	0
147.168945	153.015869	152.171143	124.204590	133.436523	...	0	0	0	0	1	0	0	0	0	0
164.145752	171.126221	165.814209	167.287109	168.461426	...	0	0	0	0	0	0	1	0	0	0
109.496338	108.715820	146.747070	138.000244	131.082275	...	0	0	0	0	0	0	1	0	0	0
165.254639	177.674805	168.733643	136.787842	159.784180	...	0	0	0	0	1	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
164.273438	173.517578	166.562744	169.823486	174.394287	...	0	0	0	0	0	0	0	0	0	1
194.697754	102.145996	67.431641	189.038330	121.057129	...	0	1	0	0	0	0	0	0	0	0
122.392334	69.797119	65.513428	111.660156	80.384766	...	0	0	0	0	0	0	0	0	0	1
128.271973	83.036377	77.726807	135.552979	152.290283	...	0	0	0	0	0	0	0	0	0	1
108.646729	92.931641	155.751953	102.204102	140.394287	...	0	1	0	0	0	0	0	0	0	0

Importantly, labels that converted from the original label set should be stored to extract the correct output, as colour and texture are calculated with two different models. Using the *concat*() function, we would append those results at the end of the dataset and remove the original colour and texture labels.

Model Design

Model is the essential part of machine learning, as the precision of the model is being trained and evaluated in this stage. This section introduces a model that is considered to be suitable for solving this question.

Logistics Regression

Logistics regression is the model that is implemented with python (Using PyCharm). The chosen model considers the fact that this question is a question of classification. We would classify the image and judge the label (Colour, Texture) of the image. The logistics regression is a model for classification. For this model, hyper-parameters are chosen and with the reason lists as follows:

- multi-class (ovr or multinomial): As we have lots of labels in colour tag and texture tag, this is a multi-classification question. We choose ovr or ovo for classification. Multinomial uses multinomial regression for prediction, and ovr divides the question into several classifiers. (For example, we have [1,2,3] as outputs, classifier of over makes the probability represents as [1], [2, 3], which is one vs rest strategy)
- solver: The calculus of optimization problem. The property can be adjusted manually.
- C: C changes the generalization performance of the model. The higher the C, the more general the model represents, but the model would be less accurate.
- max_iteration: This property defines the maximum iteration of the model. It just does not need to be adjusted and can leave it default.
- penalty: The loss function type increment the availability and generalization performance of the model.
- class_weight: The weight of classes of each model.

The model's hyperparameter is modified and adjusted to a best-match one, which could be the model that we want.

[IMPORTANT!] With data preparation, I annotated the represented line of data pre-processing. Actually, this model includes the process of data pre-processing, data validation and data training.

```
if __name__ == '__main__':
    # data_prep()
    # data_division()
    # data_norm_fold()
    model_train()
    # res_ret()
```

Linear Regression

Linear Regression is one of the linear models but is being implemented with the Conda python environment (Jupiter Notebook). The model is a ordinary least squares Linear Regression, which is a classical model that solve the linear question. The model considers the image to be linear (May represents objects within the image looks obvious). Linear regression finds an axiom in the case of summarizing the motion of scatters. The calculus of the model may represent as follows:

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$$

Linear regression may considers the output to be numbers instead of labels, so we uses one-hot encoding to encode our output, and reverse our encoding in our final step. Since the output of the model would represents as a set of possibilities, we choose the highest possibility and set to one, and set the rest of labels to 0.

Reverse from the one-hot encoding is implemented as follows:

```
[black, white] -> [[1, 0], [0, 1]]
if output = [1, 0]
label = labels[[[1, 0], [0, 1]].find(output)]
```

Random Forest

Random Forest is one of the Ensemble learning models which is being implemented with the Conda python environment (Jupiter Notebook). Ensemble models represents its advantages with solving the problem of a single model may performs not quite well. According to the random forest documentation, the random forest creates a number of decision trees that any of the decision trees is unrelated to each other. Each decision tree will create a result representing the decision tree's output and the final classification of the prediction based on the number of decision tree

voted to the label. Briefly, I consider the decision tree represents its calculus as follows:

- Train a decision tree with properties using the sampled dataset
- Choose a random property as splitting property of the decision tree
- Repeat the second step until any node of the decision tree is completely independent
- Repeat step 1-3 to create a variety of decision trees.

The random forest allows the following attributes being judged:

- n_estimator: This decides how many decision trees are generated.
- criterion: The evaluation of how well properties are being split with the decision tree.
- min_sample_split: This evaluates the minimum number of samples required to split an internal node.
- min_weight_fraction_leaf: This evaluates minimum weighted fraction of the sum total of weights required to be at a leaf node.
- min_impurity_decrease: A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
- oob_score: Use out-of-bag samples to increase the generalize property of the model.
- random_state: The randomness when building the tree.
- ccp_alpha: Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen.

A random forest calculates a variety of features and automatically allows for feature selection; moreover, the random forest balances the unbalanced dataset and keeps its accuracy when many features are missing.

Model validation

Accuracy identification

Accuracy Score

Accuracy score calculates the rate of the model returning true output. This tells about the distribution of responses. Calculus for accuracy score is described as follows:

$$Accuracy = \frac{n_{correct}}{n_{total}}$$

Balanced Accuracy Score

Balanced accuracy score is considered to solve the issue of imbalanced dataset is not evaluated precisely with accuracy score. Balanced accuracy score can be summarized as follows:

$$Balanced = \frac{TPR + TNR}{2}$$

As TPR is the true positive rate and TNR is the true negative rate

Confusion Matrix

Confusion matrix evaluates the number of conditions that elements of the model is same as the predicted model. For example, if we predicts with [a, b, c, c] but we get [a, c, c, c] for true, we have the confusion matrix with

```
[
1,0,0
0,0,0
0,1,2
]
```

K-Fold validation

We use k-Fold validation technology for testing the validation of the model. K-Fold technology splitting a dataset into several parts that have equal size. For each iteration, the model will use one part for validation and training with the rest of the model. The validation does helps to improve the precision rate of the model by modifying the hyper-parameter of the model. A preferred fold is to be considered $k = 5$, that the model uses k-Fold technology that included in the sklearn package splitting the dataset into five parts. The use of the k-fold validation represents as follows:

$$Average\ Accuracy\ Score = \frac{u_1 + u_2 + u_3 + u_4 + u_5}{n}$$

$$Average\ Balanced\ Accuracy\ Score = \frac{A_1 + A_2 + A_3 + A_4 + A_5}{n}$$

Whereas A represents the balanced accuracy score for each folded dataset, and u represents as the accuracy score for each folded dataset.

Implementation

The implementation for this practical includes the following:

- All datasets that cleaned
- The implementation of the Linear Regression and represented result
- The implementation of the Logistics Regression model and represented result
- The implementation of the Random Forest model and represented result

Testing

Validation test for each model represents as follows:

Linear Regression

```
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains classes not in y_true
  warnings.warn('y_pred contains classes not in y_true')
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains classes not in y_true
  warnings.warn('y_pred contains classes not in y_true')
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains classes not in y_true
  warnings.warn('y_pred contains classes not in y_true')
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains classes not in y_true
  warnings.warn('y_pred contains classes not in y_true')
```

```
Color model evaluation: Accuracy - 0.4157303370786517, Balanced Accuracy - 0.18395061728395062
Texture model evaluation: Accuracy - 0.449438202247191, Balanced Accuracy - 0.31107625482625484
Color model evaluation: Accuracy - 0.3707865168539326, Balanced Accuracy - 0.17898242630385486
Texture model evaluation: Accuracy - 0.3970037453183521, Balanced Accuracy - 0.29313814280046235
```

```
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains classes not in y_true
  warnings.warn('y_pred contains classes not in y_true')
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains classes not in y_true
  warnings.warn('y_pred contains classes not in y_true')
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains classes not in y_true
  warnings.warn('y_pred contains classes not in y_true')
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains classes not in y_true
  warnings.warn('y_pred contains classes not in y_true')
```

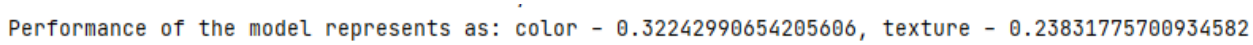
```
Color model evaluation: Accuracy - 0.36704119850187267, Balanced Accuracy - 0.29540818089311965
Texture model evaluation: Accuracy - 0.39325842696629215, Balanced Accuracy - 0.22603309688505951
Color model evaluation: Accuracy - 0.36704119850187267, Balanced Accuracy - 0.19134350851640572
Texture model evaluation: Accuracy - 0.41198501872659177, Balanced Accuracy - 0.33209886739298505
Color model evaluation: Accuracy - 0.39097744360902253, Balanced Accuracy - 0.2725765612155396
Texture model evaluation: Accuracy - 0.4699248120300752, Balanced Accuracy - 0.34875412286126567


Overall evaluation:
Color Accuracy: 0.38231533890907043, Color balanced accuracy: 0.22445225884257408
Texture Accuracy: 0.42432204105770044, Texture balanced accuracy: 0.30222009695320545
```

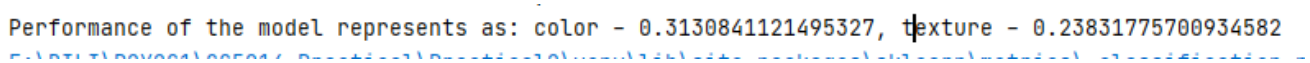
```
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains classes not in y_true
  warnings.warn('y_pred contains classes not in y_true')
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains classes not in y_true
  warnings.warn('y_pred contains classes not in y_true')
```

As we can see above, Linear regression model represents some better than the logistics model, but the accuracy is still not satisfiable.

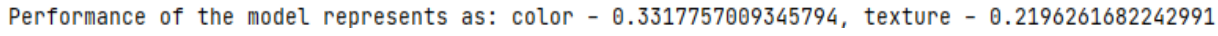
Logistics Regression



This figure represents the accuracy of logistics regression with C=0.1, multiclass=Multinomial, penalty=l2 and solver=newton-cg.



This figure represents the accuracy of logistics regression with C=1, multiclass=Multinomial, penalty=l2 and solver=newton-cg. Max iteration is set 5000.



This figure represents the accuracy of logistics regression with C=1, multiclass=ovr, penalty=l2 and solver=newton-cg.

As we can see above, changing parameters may represents less changes to the performance of the model. As a result, Logistics regression may represents as a less effective model for this practical.

Random Forest

The figure showing below represents the performance of the model in validation stage. Random forest represents a higher accuracy than both linear regression model and logistics models, so the random forest might be a better model for the practical.

```
color 0.5730337078651685
texture 0.5692883895131086
```

```
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains c
  warnings.warn('y_pred contains classes not in y_true')
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains c
  warnings.warn('y_pred contains classes not in y_true')
```

```
color 0.5093632958801498
texture 0.5318352059925093
```

```
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains c
  warnings.warn('y_pred contains classes not in y_true')
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains c
  warnings.warn('y_pred contains classes not in y_true')
```

```
color 0.4606741573033708
texture 0.49063670411985016
```

```
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains c
  warnings.warn('y_pred contains classes not in y_true')
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains c
  warnings.warn('y_pred contains classes not in y_true')
```

```
color 0.5468164794007491
texture 0.5767790262172284
```

```
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains c
  warnings.warn('y_pred contains classes not in y_true')
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains c
  warnings.warn('y_pred contains classes not in y_true')
```

```
color 0.4774436090225564
texture 0.5488721804511278


Overall evaluation:
Color Accuracy: 0.5134662498943989, Color balanced accuracy: 0.4574044979752355
Texture Accuracy: 0.543482301258765, Texture balanced accuracy: 0.605707987157318
```

Final testing.

The image showing below represents the overall performance for each models, which looks less satisfiable:

2 days ago	0.16121	Logistic Regression
12 hours ago	0.1643	Linear Regression
9 minutes ago	0.15949	Random Forest - n_estimator = 10
7 minutes ago	0.15772	Random Forest - n_estimator = 60
4 minutes ago	0.16918	Random Forest - n_estimator = 120

Image for color recognition

2 days ago	0.22817
12 hours ago	0.26062
8 minutes ago	0.24166
7 minutes ago	0.22751
4 minutes ago	0.22722

Image for texture recognition, reflected models represents as above accordingly

Unfortunately, random forest represents not such satisfiable than it was done in the validation process, which may means that some features leads to overfitting of the model. We may need to identify which feature is strongly related to the model and may remove the feature in next step.

EXT. Accuracy for random forest have increased a bit. Image above does not represents the actual final result of the testing.

Evaluation

Linear regression and Logistics regression models

These two models performs not quite well in both validation step and testing step. This may caused by object identification is not a regression question, and hence is not able to be classified by Logistics Regression.

Random Forest

When using k-fold strategy testing the performance of the model,

```
color 0.5730337078651685
texture 0.5692883895131086

D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains c
  warnings.warn('y_pred contains classes not in y_true')
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains c
  warnings.warn('y_pred contains classes not in y_true')

color 0.5093632958801498
texture 0.5318352059925093

D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains c
  warnings.warn('y_pred contains classes not in y_true')
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains c
  warnings.warn('y_pred contains classes not in y_true')

color 0.4606741573033708
texture 0.49063670411985016

D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains c
  warnings.warn('y_pred contains classes not in y_true')
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains c
  warnings.warn('y_pred contains classes not in y_true')

color 0.5468164794007491
texture 0.5767790262172284

D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains c
  warnings.warn('y_pred contains classes not in y_true')
D:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1814: UserWarning: y_pred contains c
  warnings.warn('y_pred contains classes not in y_true')

color 0.4774436090225564
texture 0.5488721804511278


Overall evaluation:
Color Accuracy: 0.5134662498943989, Color balanced accuracy: 0.4574044979752355
Texture Accuracy: 0.543482301258765, Texture balanced accuracy: 0.605707987157318
```

We can find out that Random tree performs well in validation stage, but is not generalized. I considers the issue might be leads by noises of pictures: Objects might be hidden or recognition of the object might be interrupted.

As a result, the random forest represents overfit when predicting models with the test set. With analyzing the dataset, this might be caused by noises from images. As a result, the random forest performs not quite well with this dataset.

Conclusion

As a result, random forest represents the best performance in object recognition, but causes overfitting in final testing. However, I believe that the random forest model represents a better performance with object identification in this practical. A further work for this practical is to do feature selection furtherly, as remove some highly related labels that may improve the performance of the model.

This practical let me learnt a lot. A further study of machine learning may focus on feature selection, as to find some more ways of identify features in the dataset and training with model with a better and more suitable dataset.