210801159

April 23, 2023

1 Glass Classification

Assignment submitted by : ***Ritvik Prasad M***

Registration Number : **210801159**

1.0.1 Context and Content

Context

This is a Glass Identification Data Set from UCI. It contains 9 attributes.

The response is glass type(discrete 7 values)

Kaggle Link

UCI dataset Link

Content

Attribute Information

- RI: refractive index
- Na: Sodium (unit measurement: weight percent in corresponding oxide, as are attributes 4-10)
- Mg: Magnesium
- Al: Aluminum
- Si: Silicon
- K: Potassium
- Ca: Calcium
- Ba: Barium
- Fe: Iron
- Type of glass: (class attribute)
  - 1 building_windows_float_processed

- 2 building_windows_non_float_processed
- 3 vehicle_windows_float_processed
- 4 vehicle_windows_non_float_processed (none in this database)
- 5 containers
- 6 tableware
- 7 headlamps

1.0.2 Data Loading and Analysis

[1]:
```python
# importing essential libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[74]:
```python
# loading the dataset
df = pd.read_csv("./glass.csv")
df.head()
```

[74]:

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	1

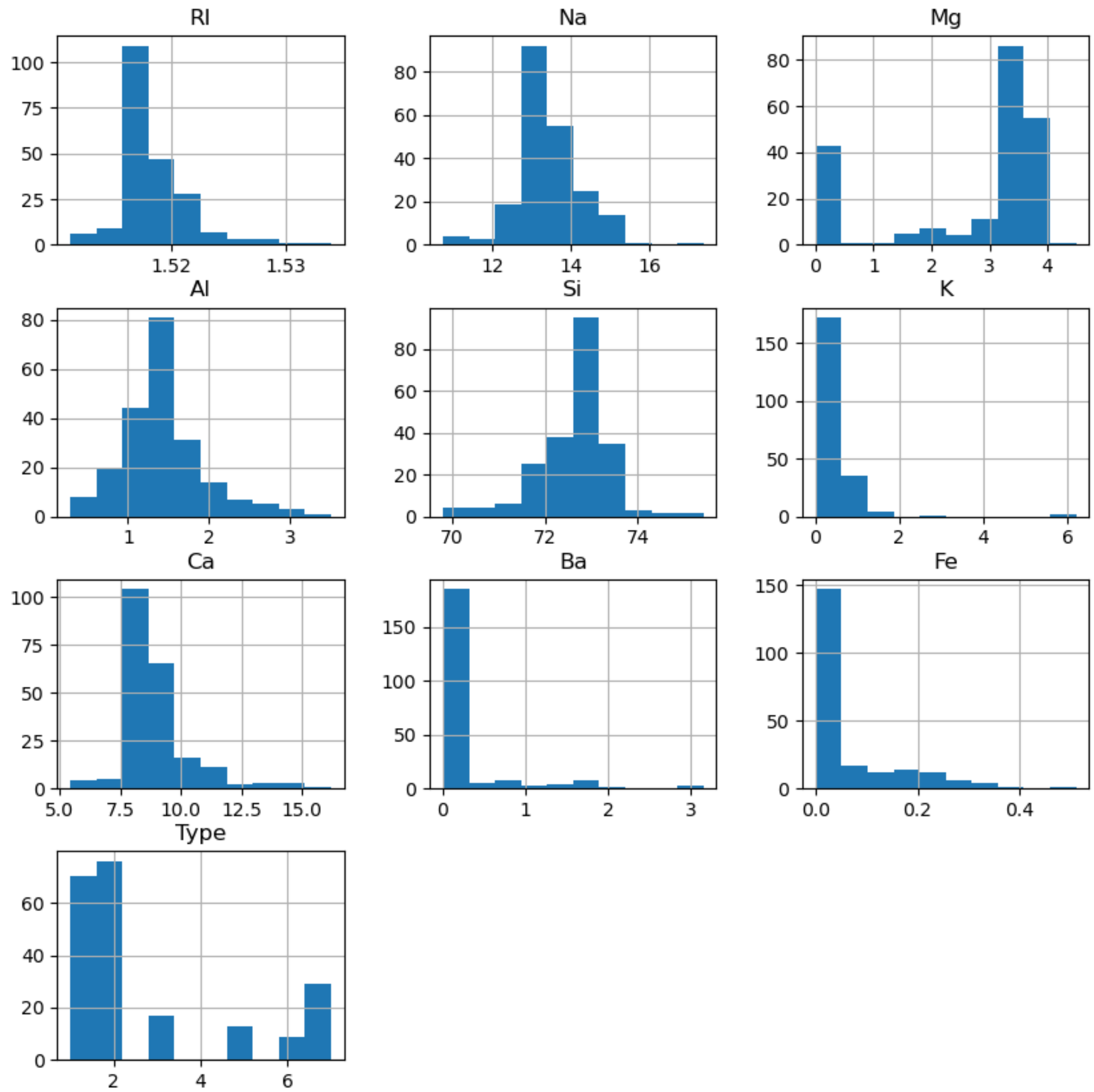
[3]:
```python
#getting information about the data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   RI      214 non-null    float64
 1   Na      214 non-null    float64
 2   Mg      214 non-null    float64
 3   Al      214 non-null    float64
 4   Si      214 non-null    float64
 5   K       214 non-null    float64
 6   Ca      214 non-null    float64
 7   Ba      214 non-null    float64
 8   Fe      214 non-null    float64
 9   Type    214 non-null    int64
```

```
dtypes: float64(9), int64(1)
memory usage: 16.8 KB
```

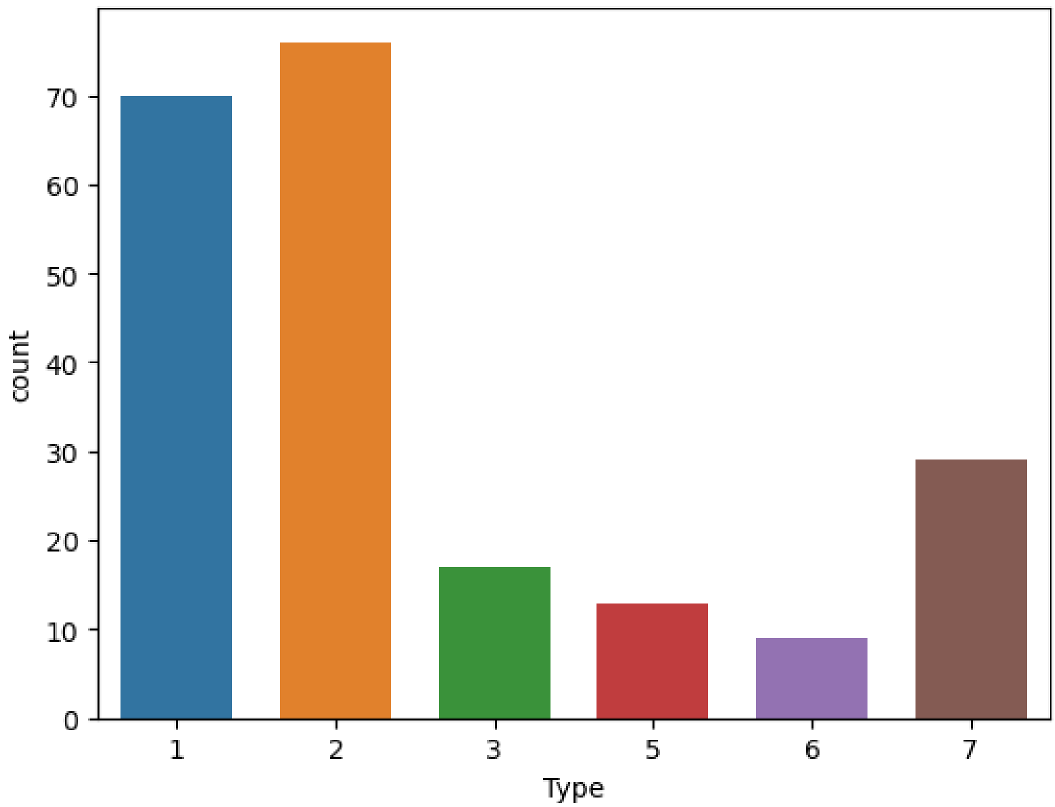
[4]:

```python
# understanding the correlation between the data
df.hist(figsize=(10,10));
```



[5]:

```python
# analyzing the number and types of outputs
sns.countplot(x=df["Type"], width=0.7);
```



[6]:
```
# finding the total number of possible outcomes
df.Type.unique(), len(df.Type.unique())
```

[6]:
```
(array([1, 2, 3, 5, 6, 7]), 6)
```

Inference

As the data outcomes are discrete, we would use a multiclass classification algorithm such as non-binary multiclass classification.

For this data, I will use **Random Forest Classifier** Algorithm.

1.0.3 Model Creation and Training

[7]:
```
# splitting the data into labels and outcomes
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
X.shape , y.shape
```

[7]:
```
((214, 9), (214,))
```

[8]:
```
# splitting the data into train and test set with test size of 0.2 as the␣
↪availability of data is less
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[9]:
```
# importing necessary libraries for model creation
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
```

[10]:
```
# creating the pipeline for the model and getting the pipeline paramerters
pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("randomForest", RandomForestClassifier())
])
pipe.get_params()
```

[10]:
```
{'memory': None,
 'steps': [('scaler', StandardScaler()),
  ('randomForest', RandomForestClassifier())],
 'verbose': False,
 'scaler': StandardScaler(),
 'randomForest': RandomForestClassifier(),
 'scaler__copy': True,
 'scaler__with_mean': True,
 'scaler__with_std': True,
 'randomForest__bootstrap': True,
 'randomForest__ccp_alpha': 0.0,
 'randomForest__class_weight': None,
 'randomForest__criterion': 'gini',
 'randomForest__max_depth': None,
 'randomForest__max_features': 'sqrt',
 'randomForest__max_leaf_nodes': None,
 'randomForest__max_samples': None,
 'randomForest__min_impurity_decrease': 0.0,
 'randomForest__min_samples_leaf': 1,
 'randomForest__min_samples_split': 2,
 'randomForest__min_weight_fraction_leaf': 0.0,
 'randomForest__n_estimators': 100,
 'randomForest__n_jobs': None,
 'randomForest__oob_score': False,
 'randomForest__random_state': None,
 'randomForest__verbose': 0,
 'randomForest__warm_start': False}
```

[11]:
```
# creating a cross validation estimator grid with estimators and maximum depth␣
↪as grid parameters

estimator = GridSearchCV(estimator=pipe, param_grid={
    'randomForest__n_estimators':[i for i in range(50,150,5)],
    'randomForest__max_depth':[i for i in range(1,11)]
}, scoring='accuracy', cv=3)
```

[52]:
```
# fitting the estimator
estimator.fit(X_train, y_train)
```

[52]:
```
GridSearchCV(cv=3,
             estimator=Pipeline(steps=[('scaler', StandardScaler()),
                                       ('randomForest',
                                        RandomForestClassifier())]),
             param_grid={'randomForest__max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9,
                                                     10],
                         'randomForest__n_estimators': [50, 55, 60, 65, 70, 75,
                                                        80, 85, 90, 95, 100,
                                                        105, 110, 115, 120, 125,
                                                        130, 135, 140, 145]},
             scoring='accuracy')
```

[53]:
```
# storing the results of the estimator in a dataframe
result_df = pd.DataFrame(estimator.cv_results_)
```

[54]:
```
# getting the parameters for the best optimum model
print(estimator.best_index_)
print(estimator.best_params_)
print(estimator.best_score_)
```

```
125
{'randomForest__max_depth': 7, 'randomForest__n_estimators': 75}
0.7543859649122807
```

[55]:
```
# getting the details about best optimum model from result database
result_df.iloc[150]
```

[55]:
```
mean_fit_time
0.122424
std_fit_time
0.002658
mean_score_time
0.010924
std_score_time
0.001117
param_randomForest__max_depth
```

```
8
param_randomForest__n_estimators
100
params                                   {'randomForest__max_depth': 8,
'randomForest__…
split0_test_score
0.578947
split1_test_score
0.842105
split2_test_score
0.701754
mean_test_score
0.707602
std_test_score
0.107513
rank_test_score
93
Name: 150, dtype: object
```

[56]:
```
# creating an improvised estimator with parameters close to best estimate of␣
 ↪previous estimator
est_depth = estimator.best_params_['randomForest__max_depth']
est_estimators = estimator.best_params_['randomForest__n_estimators']
delta1 = 5
delta2 = 2
estimator_improvised = GridSearchCV(estimator=pipe, param_grid={
    'randomForest__n_estimators':[i for i in␣
 ↪range(est_estimators-delta1,est_estimators+delta1+1)],
    'randomForest__max_depth':[i for i in␣
 ↪range(est_depth-delta2,est_depth+delta2+1)],
}, scoring='accuracy', cv=3)
```

[57]:
```
# fitting the improvised estimator
estimator_improvised.fit(X_train, y_train)
```

[57]:
```
GridSearchCV(cv=3,
             estimator=Pipeline(steps=[('scaler', StandardScaler()),
                                       ('randomForest',
                                        RandomForestClassifier())]),
             param_grid={'randomForest__max_depth': [5, 6, 7, 8, 9],
                         'randomForest__n_estimators': [70, 71, 72, 73, 74, 75,
                                                        76, 77, 78, 79, 80]},
             scoring='accuracy')
```

[58]:
```
# storing the results of improvised estimator in a dataframe
result2_df = pd.DataFrame(estimator.cv_results_)
```

[59]:
```
# getting the parameters for the best optimum model from the improvised␣
↪estimator
print(estimator_improvised.best_index_)
print(estimator_improvised.best_params_)
print(estimator_improvised.best_score_)
```

```
27
{'randomForest__max_depth': 7, 'randomForest__n_estimators': 75}
0.7485380116959064
```

[60]:
```
# getting the details about best optimum model from improvised result database
result2_df.iloc[181]
```

[60]:
```
mean_fit_time
0.071356
std_fit_time
0.000997
mean_score_time
0.008694
std_score_time
0.000641
param_randomForest__max_depth
10
param_randomForest__n_estimators
55
params                                   {'randomForest__max_depth': 10,
'randomForest_…
split0_test_score
0.596491
split1_test_score
0.894737
split2_test_score
0.684211
mean_test_score
0.725146
std_test_score
0.125152
rank_test_score
30
Name: 181, dtype: object
```

[67]:
```
# getting the top 2 estimators
model1 = estimator.best_estimator_
model2 = estimator_improvised.best_estimator_
```

1.0.4 Model Evaluation

[68]:
```python
# comparing the top two models

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

preds1 = model1.predict(X_test)
preds_train1 = model1.predict(X_train)
accuracy_test1 = accuracy_score(y_true=y_test, y_pred=preds1)
accuracy_train1 = accuracy_score(y_true=y_train, y_pred=preds_train1)
cmatrix1 = confusion_matrix(y_true=y_test, y_pred=preds1)
print(f'Model 1 Metrics')
print(f'accuracy (test)       : {accuracy_test1}\n\tin precentage :
 ↪\t{accuracy_test1*100} %\n')
print(f'accuracy (train)      : {accuracy_train1}\n\tin precentage :
 ↪\t{accuracy_train1*100} %\n')
print(f'confusion matrix (for 6 outcomes):\n{cmatrix1}')

preds2 = model2.predict(X_test)
preds_train2 = model2.predict(X_train)
accuracy_test2 = accuracy_score(y_true=y_test, y_pred=preds2)
accuracy_train2 = accuracy_score(y_true=y_train, y_pred=preds_train2)
cmatrix2 = confusion_matrix(y_true=y_test, y_pred=preds2)
print(f'\nModel 2 Metrics')
print(f'accuracy (test)       : {accuracy_test2}\n\tin precentage :
 ↪\t{accuracy_test2*100} %\n')
print(f'accuracy (train)      : {accuracy_train2}\n\tin precentage :
 ↪\t{accuracy_train2*100} %\n')
print(f'confusion matrix (for 6 outcomes):\n{cmatrix2}')
```

```
Model 1 Metrics
accuracy (test)       : 0.9069767441860465
        in precentage : 90.69767441860465 %

accuracy (train)      : 0.9766081871345029
        in precentage : 97.6608187134503 %

confusion matrix (for 6 outcomes):
[[11  0  0  0  0  0]
 [ 3 11  0  0  0  0]
 [ 1  0  2  0  0  0]
 [ 0  0  0  4  0  0]
 [ 0  0  0  0  3  0]
 [ 0  0  0  0  0  8]]

Model 2 Metrics
accuracy (test)       : 0.813953488372093
```

```
        in precentage : 81.3953488372093 %

accuracy (train)      : 0.9766081871345029
        in precentage : 97.6608187134503 %

confusion matrix (for 6 outcomes):
[[11  0  0  0  0  0]
 [ 3 10  0  0  0  1]
 [ 1  1  1  0  0  0]
 [ 0  2  0  2  0  0]
 [ 0  0  0  0  3  0]
 [ 0  0  0  0  0  8]]
```

[75]:
```
print(f'Test delta  : {(accuracy_test1-accuracy_test2)*100}%')
print(f'Train delta : {(accuracy_train1-accuracy_train2)*100}%')
```

```
Test delta  : 9.302325581395344%
Train delta : 0.0%
```

We can see the metrics as:

- Model1
  - Test Accuracy : 90.69%
  - Train Accuracy : 97.66%
- Model2
  - Test Accuracy : 81.39%
  - Train Accuracy : 97.66%

We can see that *model1* perfoms better on train set than test set by a margin of **9.30%** and *model1* perfoms better on test set than train set by a margin of **0.00%**.

Hence, from this observation, we can say that model1 outperfoms model2.

Therefore model1 is the most optimum model.

1.0.5 Exporting the Model

[71]:
```
# exporting the model in .joblib format
from joblib import dump
dump(model1, "glass_pred.joblib")
```

[71]: ['glass_pred.joblib']

[73]:
```
# loading the model and finding its accuracy

from joblib import load
from sklearn.metrics import accuracy_score
```

```
load_model = load("./glass_pred.joblib")

pred_load = load_model.predict(X_test)
accuracy_load_model = accuracy_score(y_true=y_test,y_pred=pred_load)
print(f'Accuracy of the model loaded : {accuracy_load_model*100} %')
```

```
Accuracy of the model loaded : 90.69767441860465 %
```

1.0.6 Conclusion

Hence, a **Random Forest Classifier** algorithm is implemented on the given data and a Machine Learning model with **90.698% accuracy** is obtained.