20250427_01

April 27, 2025

Today's objective : classify wines into 3 categories with Logistic Regression and Decision Tree.

[93]:

```
# Loading the data and check how many are there
import pandas as pd
from sklearn.datasets import load_wine

data = load_wine()
X = pd.DataFrame(data.data, columns = data.feature_names)
y = pd.Series(data.target)

print(X.shape)
print(y.value_counts())
```

```
(178, 13)
1    71
0    59
2    48
Name: count, dtype: int64
```

X and y has the same number of rows, nice.

[95]:

```
# Take a look at the dataset
X.describe()
```

[95]:

```
          alcohol  malic_acid         ash  alcalinity_of_ash   magnesium  \
count  178.000000  178.000000  178.000000         178.000000  178.000000
mean    13.000618    2.336348    2.366517          19.494944   99.741573
std      0.811827    1.117146    0.274344           3.339564   14.282484
min     11.030000    0.740000    1.360000          10.600000   70.000000
25%     12.362500    1.602500    2.210000          17.200000   88.000000
50%     13.050000    1.865000    2.360000          19.500000   98.000000
75%     13.677500    3.082500    2.557500          21.500000  107.000000
max     14.830000    5.800000    3.230000          30.000000  162.000000

       total_phenols  flavanoids  nonflavanoid_phenols  proanthocyanins  \
count     178.000000  178.000000            178.000000       178.000000
mean        2.295112    2.029270              0.361854         1.590899
std         0.625851    0.998859              0.124453         0.572359
min         0.980000    0.340000              0.130000         0.410000
```

```
25%         1.742500    1.205000              0.270000         1.250000
50%         2.355000    2.135000              0.340000         1.555000
75%         2.800000    2.875000              0.437500         1.950000
max         3.880000    5.080000              0.660000         3.580000

       color_intensity         hue  od280/od315_of_diluted_wines      proline
count       178.000000  178.000000                    178.000000   178.000000
mean          5.058090    0.957449                      2.611685   746.893258
std           2.318286    0.228572                      0.709990   314.907474
min           1.280000    0.480000                      1.270000   278.000000
25%           3.220000    0.782500                      1.937500   500.500000
50%           4.690000    0.965000                      2.780000   673.500000
75%           6.200000    1.120000                      3.170000   985.000000
max          13.000000    1.710000                      4.000000  1680.000000
```

X has no missing values, nice. But the value of each features varies a lot, need to standardize (for Logistic Regression).

[97]:

```python
# Making pipelines (Logistic Regression and Decision Tree)
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

logreg_pipeline = Pipeline([('scaler', StandardScaler()), ('logreg',␣
 ↪LogisticRegression(solver = 'liblinear', random_state = 42))])

tree_pipeline = Pipeline([('tree', DecisionTreeClassifier(random_state = 42))])
```

Now we train and test.

[99]:

```python
# Making training and testing sets
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)
```

[107]:

```python
# Training the models
from sklearn.model_selection import GridSearchCV

logreg_parm_grid = {'logreg__C':[0.01, 0.1, 1, 10, 100]}
logreg_grid = GridSearchCV(logreg_pipeline, logreg_parm_grid, cv = 5)
logreg_grid.fit(X_train, y_train)

tree_parm_grid = {'tree__max_depth': [3, 5, 10, None], 'tree__min_samples_leaf':
 ↪ [1, 3, 5]}
tree_grid = GridSearchCV(tree_pipeline, tree_parm_grid, cv = 5)
tree_grid.fit(X_train, y_train)
```

[107]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('tree',
DecisionTreeClassifier(random_state=42))]),
             param_grid={'tree__max_depth': [3, 5, 10, None],
                         'tree__min_samples_leaf': [1, 3, 5]})
```

[128]:
```
# Checking which one is the best and how best is it for each model
logreg_cv_results = pd.DataFrame(logreg_grid.cv_results_)
logreg_cv_results[['param_logreg__C', 'mean_test_score', 'std_test_score',␣
 ↪'rank_test_score']]
```

[128]:
```
   param_logreg__C  mean_test_score  std_test_score  rank_test_score
0             0.01         0.957635        0.026864                5
1             0.10         0.978818        0.017301                3
2             1.00         0.985961        0.017199                1
3            10.00         0.979064        0.017100                2
4           100.00         0.965025        0.021817                4
```

[117]:
```
tree_cv_results = pd.DataFrame(tree_grid.cv_results_)
tree_cv_results[['param_tree__max_depth', 'param_tree__min_samples_leaf',␣
 ↪'mean_test_score', 'std_test_score', 'rank_test_score']]
```

[117]:
```
   param_tree__max_depth  param_tree__min_samples_leaf  mean_test_score  \
0                      3                             1         0.922414
1                      3                             3         0.908374
2                      3                             5         0.900985
3                      5                             1         0.915271
4                      5                             3         0.915517
5                      5                             5         0.900985
6                     10                             1         0.915271
7                     10                             3         0.915517
8                     10                             5         0.900985
9                   None                             1         0.915271
10                  None                             3         0.915517
11                  None                             5         0.900985

    std_test_score  rank_test_score
0         0.014819                1
1         0.017566                8
2         0.042337                9
3         0.018323                5
4         0.017057                2
5         0.042337                9
6         0.018323                5
7         0.017057                2
8         0.042337                9
9         0.018323                5
```

```
10         0.017057                      2
11         0.042337                      9
```

Since the best params for Logistic Regression(0.985961) outperform the best of Decision Tree(0.922414), I decide to use Logisitic Regression for the final test set.

[134]:

```
# Predicting and evaluating the accuracy
best_logreg = logreg_grid.best_estimator_
test_score = best_logreg.score(X_test, y_test)

print("Test set accuracy:", test_score)
```

```
Test set accuracy: 1.0
```

The accuracy is 1.0, very nice.