26/04/22 10:06pm

*******

# Ensemble Learning

Ex :- Avengers.

# Boosting Technique :- it is methodology, not a m.L Algorithm, it is applied to an Existing m.L, mostly applied on Decision Tree.

Ex :- Student

* Exam

⇒ not prepared well, Focus on wrong answers only

* Attempt Exam again

⇒ rectified but missed another section

* Again Exam

⇒ Total correctly done.

* Boosting :-

⇒ Decision Tree 1 :- misclassified records

⇒ Decision Tree 2 :- misclassified records

⇒ Decision Tree 3 :-

* 2nd Weak learner is dependent on 1st weak learner

↓

algorithm doesn't give more accuracy.

* Formula for Boosting :-

$$F_T(x) = \sum_{t=1}^{T} f_t(x) \rightarrow f_t(x) = \alpha_t \, h(x)$$

Summation of t=1 to T

Learning rate

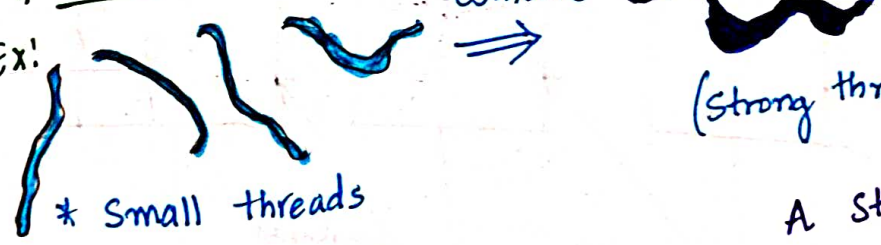
$h(x)$ → [Each model]

* boosting is the Combination of weak learners.

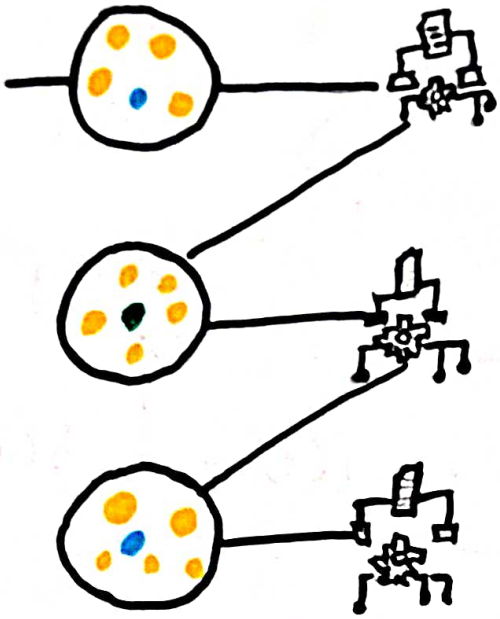
* Implies that Combination of Estimators (models) with an applied Coefficient Could act as an effective Ensemble Estimator.

# Boosting

Ex:




* Multiple Weak classifer (Stumps)



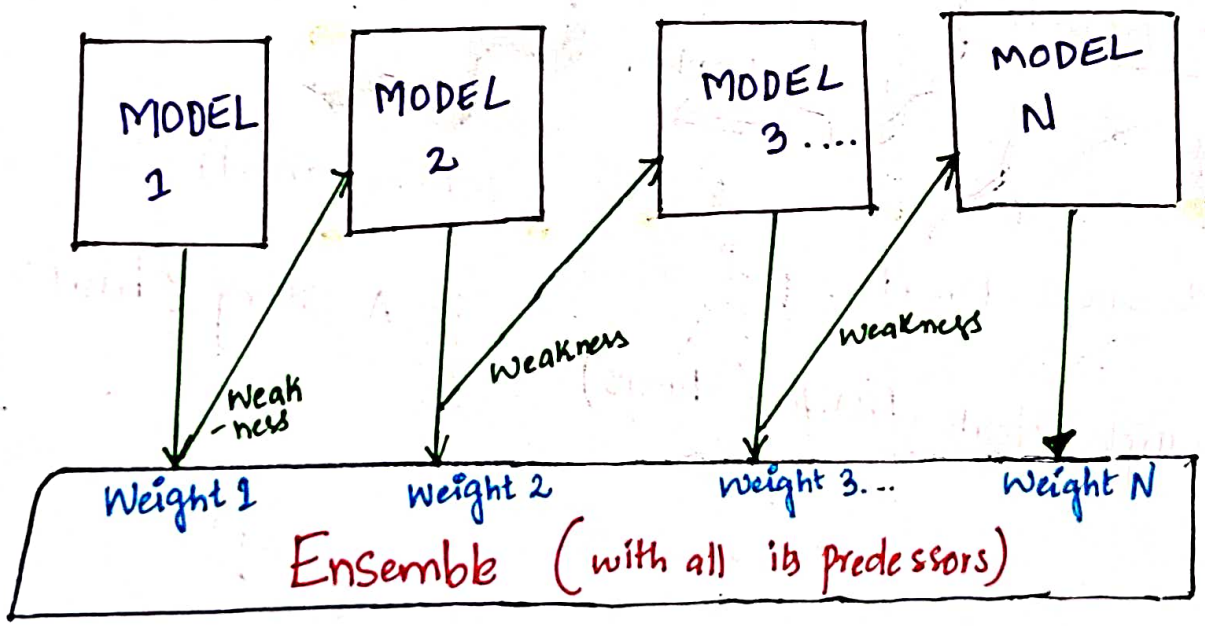
"Sequential learning":

* From the first algorithm whatever mistakes was done
* we share to the second algorithm.
* again it shared to third algorithm. - - -
* it is in "Sequential Order"

# Boosting - "Sequential" Learning.

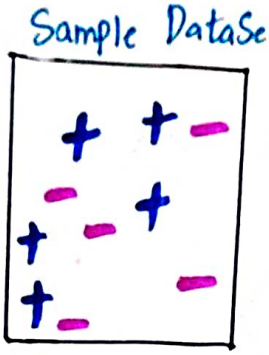
1. Ada Boost → Boosting.
   - Adaptive

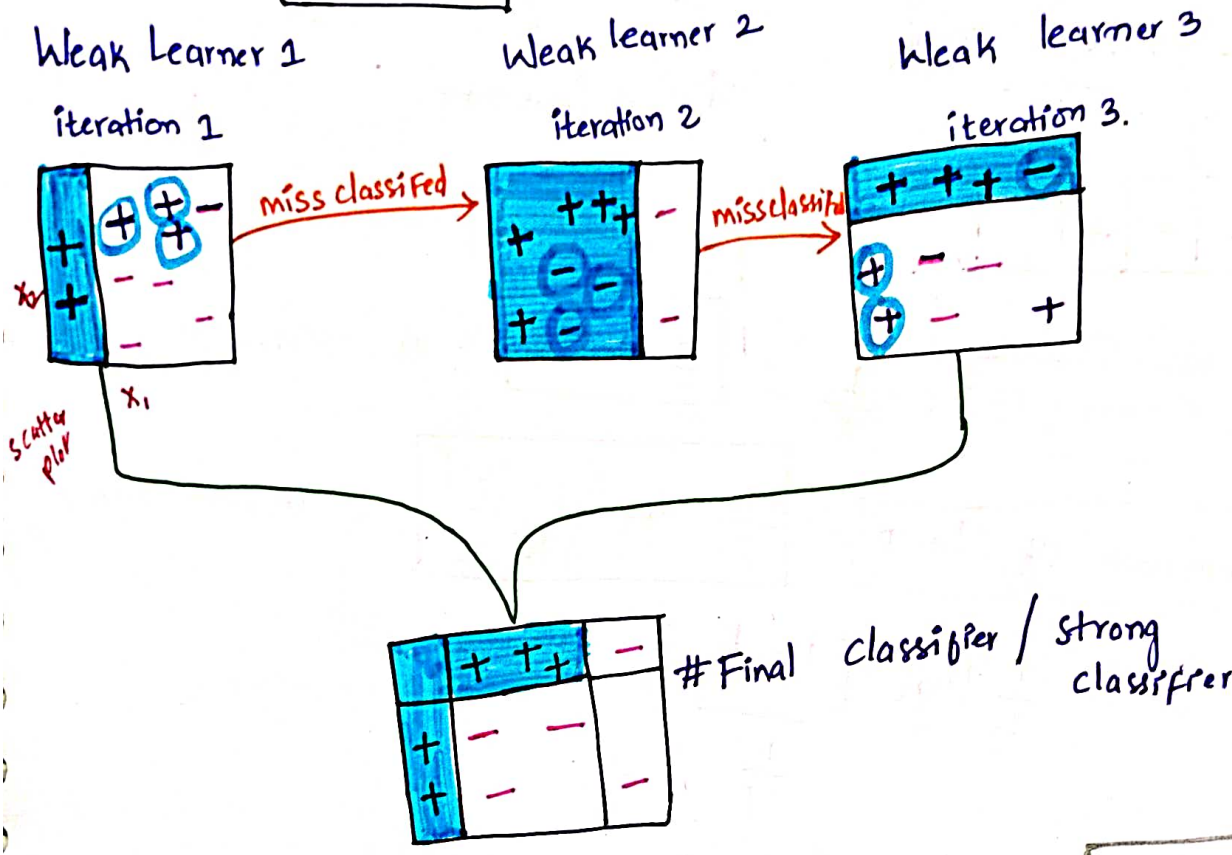
Ex:- Model, 1, 2, ... N are individual models (eg. Decision Tree)



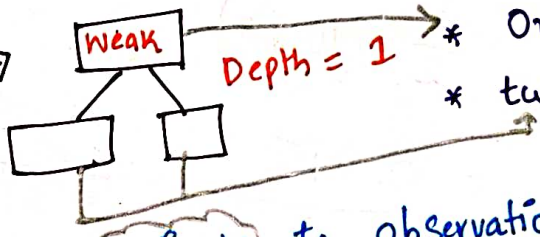
covid (positive / negative)

Ex:-




* In Adaboost :- Each weak learner is called "stump"

* Stump ⇒ 


* One root node
* two Leaf nodes.

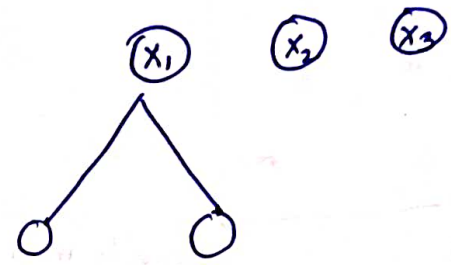
* Step 1 :- Adaboost assigns weights to observations which are incorrectly predicted

Step 2 :- Next model works to predict these values correctly which are wrongly predicted before.

Ex:-

# Taken by Entropy / Information gain

| | $X_1$ | $X_2$ | $X_3$ | $X_4$ | O/p | weights |
|---|---|---|---|---|---|---|
| 1 | – | – | – | – | Yes | 1/7 ✓ |
| 2 | – | – | – | – | No | 1/7 ✓ |
| 3 | – | – | – | – | | 1/7 ✓ |
| X 4 | – | – | – | – | | 1/7 X |
| 5 | – | – | – | – | | 1/7 ✓ |
| 6 | – | – | – | – | | 1/7 ✓ |
| 7 | – | – | – | – | | 1/7 ✓ |




{weak Learner}

*1. **Total Error [TE]** :- $\frac{1}{7}$ :- $\varepsilon_t = \varepsilon_w$ ∴ $\varepsilon_t$ = "Episinol" T

2. **Performance of Stump** :- $\frac{1}{2}\log_e\left[\frac{1-TE}{TE}\right]$ ∴ TE = Total Error

$$= \frac{1}{2}\log_e\left[\frac{1-\frac{1}{7}}{\frac{1}{7}}\right]$$

$$= \frac{1}{2}\log_e\left[\frac{6/7}{1/7}\right]$$

$$= \frac{1}{2}\log_e(6)$$

$$= 0.895$$

**New Sample weight** = $\text{Weight} * e^{-Ps}$ → correct records ∴ Ps = Performance Stump

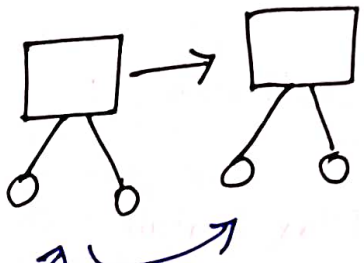
$$= \frac{1}{7} * e^{-0.895} = 0.05$$

+ **incorrect record** = $\text{weight} * e^{Ps}$ → incorrect records.

$$= \frac{1}{7} * e^{0.895} = 0.349$$

is it 1??

| Weights | New weights | Normalized weights | Buckets |
|---|---|---|---|
| $\frac{1}{7}$ | 0.05 ÷ 0.649 | 0.077 | (0 - 0.07) |
| $\frac{1}{7}$ | 0.05 ÷ 0.649 | 0.077 | (0.07 - 0.14) |
| $\frac{1}{7}$ | 0.05 | 0.077 | (0.14 - 0.21) |
| $\frac{1}{7}$ | 0.349 | 0.537 | (0.21 - 0.747) ✓ |
| $\frac{1}{7}$ | 0.05 | 0.077 | [1.747 - 0.751] ✓ |
| $\frac{1}{7}$ | 0.05 | 0.077 | — |
| $\frac{1}{7}$ | 0.05 | 0.077 | — |
| | 0.649 | ~ 1 | |



∴ 1.537 + 0.21 = 1.747

* Steps :-

⇒ Weight = $\frac{1}{n}$

⇒ Total Error_t = $\frac{[\text{correct Prediction} - n]}{n}$

⇒ Total weighted error_t = $\frac{\text{Sum (weight (i) * Error (i))}}{\text{Sum(weight)}}$

⇒ $\alpha_t = \frac{\ln [1 - TE]}{TE\ [\text{Total Error}]}$

⇒ Weight = weight * exp ( $\alpha_t$ * Total error )

⇒ total error = 0 if ($y == y^{hat}$), else 1.

# * Parameters

* n_estimors :- no. of weak learners
* Cretierion :- Gini / Entropy
* Max_Feature :- all Feature / selected ones
* Max_depth :
* min_samples_split
* n_jobs :- System is quadcore :- no. of cores = 4
  dual core :- no. of cores = 2
  octo core : no. of cores = 8
  -1 (all cores)
* Random_state :
* Learning rate :

CODE :- AdaBoost

Data :-

This Data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in agaricus and Lepiota Family (pp 500-525). Each species in indentified as definetly edible, definitely poisonous, or of unknown edibility and not recommended. this latter class was combined with the poisinous one. the Guide clearly states that there is no simple rule for determing the edibility of a mushroom; no rule like "leaflets three, Let it be" for poisinous Oak and Ivy.


→ Mushroom

Attribute Information

1. cap_shape : bell = b, Conical = c, Convex = x, flat = f, Knobbed = k, Sunken = s

2. Cap_Surface : fibrous = f, grooves = g, Scaly = y, Smooth = s

Cap_colour : brown = n, buff = b, cinnamon = c, gray = g, green = r, pink = p, purple = u, red = e, white = w, Yellow = y.

bruises ? : bruises = t, no = F

dor : almond = a, anise = l, creoste = c, fishy = y, foul = f, musty = m, none = n, punget = p, spicy = s

6. gill-attachment : black = k, brown = n, buff = b, chocolate = h,
gray = g, green = r, orange = o, pink = p,
purple = u, red = e, white = w, yellow = y

7. gill-spacing : close = c, crowded = w, distant = d

8. gill-size : broad = b, narrow = n

1. gill-color : black = k, brown = n, buff = b, chocolate = h, grave = g
green = r, orange = o, pink = p, purple = u, red = e,
white = w, yellow = y.

. stalk-shape : enlarging = e, tapering = t

stalk-root : bulbous = b, club = c, cup = u, equal = e,
rhizomorphs = z, rooted = r, missing = ?

stalk surface-above-ring : fibrous = f, scaly = y, silky = k,
smooth = s

stalk surface-below-ring : fibrous = f, scaly = y, silky = k,
Smooth = s

stalk-colour-above-ring : brown = n, buff = b, Cinnamon = c
gray = g, orange = o, pink = p, red = e,
white = w, yellow = y.

stalk-colour-below-ring : "

veil-type : Partial = p, universal = u

veil-color : brown = n, Orange = o, white = w, Yellow = y

18. ring-number :- none = n, One = o, two = t

19. ring-type :- cobwebby = c, evanescent = e, flaring = f, large = l, none = n, Pendent = p, sheathing = s, zone = z,

20. Spore-print Colour :- black = k, brown = n, buff = b, chocolate = h, green = r, orange = o, Purple = u, white = w, Yellow = y

21. Population :- abundant = a, clustered = c, numerous = n, Scattered = s, several = v, solitary = y

22. habitat :- grasses = g, Leaves = l, meadows = m, paths = p, urban = u, waste = w, woods = d.

23. class (Output)

## Bussiness Problem

Goal here is to see if we can harness the power of machine Learning and boosting to help create not just a predective model, but general Guideline for features people should look out for when picking mushrooms.

```
# df = Pd.redd_csv("mushrooms.csv"))
# df.head()
```

| class | cap Shape | cap surface | cap color | bruises | odor | gill attachment | gill spacing | gill size | gill color | ... | Stalk surface above Ring | Stalk color below ring | Stalk color below ring | veil type | veil color | ring number | ring type | Spore print color | Population | habitat |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| p | x | s | n | t | p | f | c | n | k | | s | w | w | p | w | o | p | k | s | u |
| e | x | s | y | t | a | f | c | b | k | | s | w | w | p | w | o | p | n | n | g |
| e | b | s | w | t | l | f | c | b | n | | s | w | w | p | w | o | p | n | n | m |
| p | x | y | w | t | p | f | c | n | n | | s | w | w | p | w | o | p | k | s | u |
| e | x | s | g | f | n | f | w | b | k | | s | w | w | p | w | o | e | n | a | g |

# df.shape

Out: [8124, 23]

# df.info()

Out: Range Index : 8124 Entries, 0 to 8123

| | Column | Non null Count | Dtype. |
|---|---|---|---|
| * | class | 8124 Non null | Object |
| * | cap-shape | " | " |
| * | cap surface | " | " |
| | : | : | : |
| * | habitat | " | " |

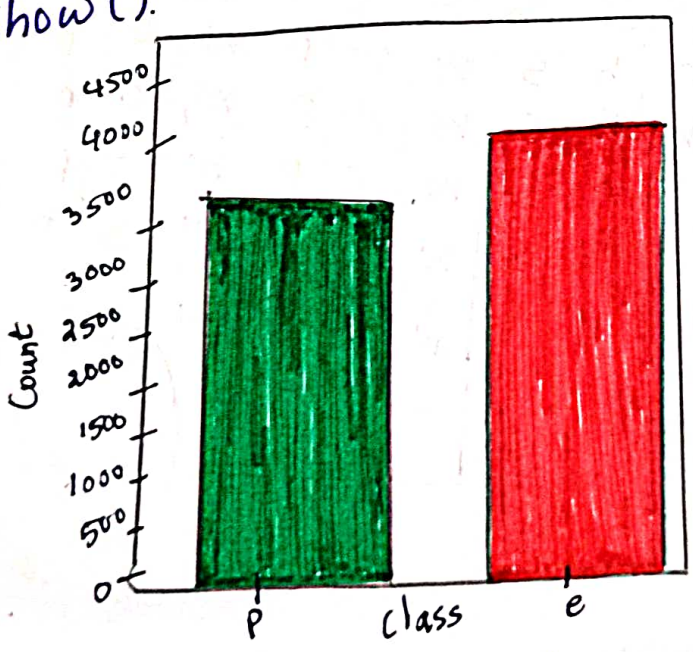
df.isnull().sum()

Out: 0.

EDA

sns.countplot(data=df, X="class", palette="Dark2")
output

plt.show().



# Transpose. gives total no. columns. without ....

```
# df.describe().transpose()
```

Out

| | Count | Unique | top | Freq |
|---|---|---|---|---|
| (o/p) Class | 8124 | 2 | e | 4208 |
| Cap Shape | 8124 | 6 | x | 3656 |
| Cap - Surface | 8124 | 4 | y | 3244 |
| Cap - color | 8124 | 10 | n | 2284 |
| bruises | 8124 | 2 | f | 4748 |
| odor | 8124 | 9 | n | 3528 |
| gill attachment | 8124 | 2 | f | 7914 |
| gill spacing | 8124 | 2 | c | 6812 |
| gill size | 8124 | 2 | b | 5612 |
| gill - color | 8124 | 12 | b | 1728 |
| Stalk - shape | 8124 | 2 | t | 4608 |
| Stalk - root | 8124 | 5 | b | 3776 |
| Stalk - Surface - above ring | 8124 | 4 | s | 5176 |
| Stalk - Surface - below ring | 8124 | 4 | s | 4936 |
| Stalk - color above ring | 8124 | 9 | w | 4464 |
| Stalk - color bebw ring | 8124 | 9 | w | 4384 |
| Veil - type | 8124 | 1 | p | 8124 |
| Veil - color | 8124 | 4 | w | 7924 |
| ring - number | 8124 | 3 | o | 7488 |
| ring - type | 8124 | 5 | p | 3968 |
| Spore - Print color | 8124 | 9 | w | 2388 |
| Population | 8124 | 6 | v | 4040 |
| habitant | 8124 | 7 | d | 3148 |

* posinous
* non posinous

## X & Y

Nomal ... (nominal data) ⇒ One hot Encoding

```
# X = pd.get_dummies(df.drop("class", axis=1), drop_first=True)

# y = df["class"]
```

# x

Out

| cap-shape_c | cap-shape_f | cap-shape_k | cap-shape_s | cap-shape_x | cap-surface_g | cap-surface_s | cap-surface_y | cap-color_c | cap-color_e | ... | population_c | population_s | population_v | population_y | habitat_g | habitat_l | habitat_m | habitat_p | habitat_u |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

8124 rows x 95 columns

# y

Out:

| | |
|---|---|
| 0 | p |
| 1 | e |
| 2 | e |
| 3 | p |
| 4 | e |

```
# x.shape, y.shape
```

Out: (8124, 95), (8124,)

## train / test split

```
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=29)
```

## MODELLING

```
from sklearn.ensemble import AdaBoostClassifier

# model = AdaBoostClassifier ()   → n_estimator = 100 (stumps)

# model.fit (x_train, y_train)
```

out : AdaBoostClassifier()

## PREDICTION

```
# ypred_train = model.predict (x_train)

# ypred_test = model.predict (x_test)
```

## Evaluation

* Accuracy

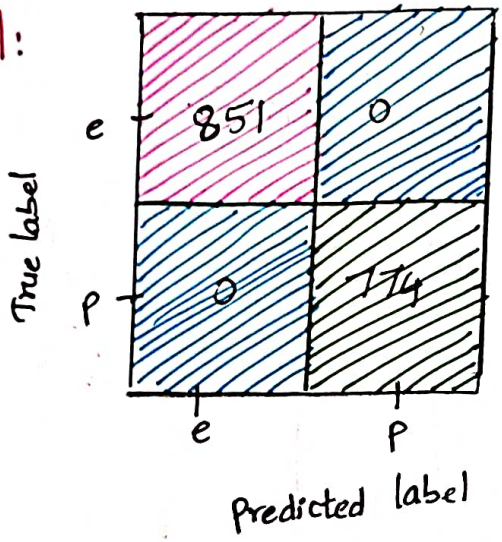
```
from sklearn.metrics import accuracy_score

# Print ("Train accuracy" : , accuracy_score (y_train, ypred_train)
# print ("Test accuray": , accuracy_score (y_test, ypred_test)
```

out:
Train Accuray : 1.0
Test Accuray : 1.0

## * Confusion Matrix

```
from sklearn.metrics import plot_confusion_matrix
# plot_confusion_matrix (model, x_test, y_test)
# plt.show()
```

Out:




## * classification Report

```
from sklearn.metrics import classification_report
# print (classification_report (y_test, y_pred_test))
```

Out:

| | Precision | recall | f1 score | Support |
|---|---|---|---|---|
| e | 1.00 | 1.00 | 1.00 | 851 |
| p | 1.00 | 1.00 | 1.00 | 774 |
| | | | 1.00 | 1625 |
| Accuracy | 1.00 | 1.00 | 1.00 | 1625 |
| macro avg | | 1.00 | 1.00 | 1625 |
| weighted Avg | 1.00 | | | |

* Cross-Validation-Score

```
from sklearn.model_selection import cross_val_score.

# scores = cross_val_score (model, x, y, cv=5)

# print (" cross validation score:", scores.mean())
```

Out : Cross validation score : 0.925

* model.feature_importances

```
# model.feature_importances_

Out  array([0. , 0. , 0. 0, 0. , 0, 0., 0.,
            0. . 0,  0. , 0.02, 0. 0.04,
            0.06, 0.02, 0.0, 0, 0.1, 0.12,
            0 ]

# f_imp = pd.DataFrame (index = X.columns, data = model.feat
                        -re = importances_ columns = ["Feature Import
                                                      ances.
```

```
# f_imp
```

Out: Feature Importances

| | |
|---|---|
| cap_shape_C | 0.00 |
| cap_shape_f | 0.00 |
| cap_shape_K | 0.00 |
| cap_shape_s | 0.00 |
| cap_shape_x | 0.00 |
| ⋮ | ⋮ |

95 rows x 1 columns

# Data Extraction (Pandas)

```
# f_imp[f_imp["feature importance"]>0]
```

Out :

| | Feature Importance |
|---|---|
| cap_color w | 0.02 |
| odor_c | 0.04 |
| odor_f | 0.04 |
| odor_n | 0.06 |
| odor_p | 0.02 |
| gill_spacingw | 0.10 |
| ⋮ | ⋮ |

## Hyper Parameter Tuning

```
from sklearn.model_selection import GridSearchCV
# estimator = AdaBoostClassifier()
# param_grid = {"n_estimators": list(range(1,101))}
# grid = GridSearchCV(estimator, param_grid, cv=5, scoring="Accuray")
# grid.fit(x_train, y_train)
# grid.best_params_
```

Out : {"n_estimator : 20}

## final model

```
# final_model = AdaBoostClassifier (n_estimators = 20)
# final_model.fit (x_train, y_train)

# Pred_train = final_model.predict (x_train)
# Pred_test = final_model.predict (x_test)

# print ("train accuray:", accuray score (y_train, Pred_train)
# print ("test accuray:", accuray score (y_test, Pred_test)
```

Out: train accuray : 1.0
Test accuray : 1.0

25/04/22
10:00pm

```
# final model.feature_importances_
```

Out: array ([0. , 0. , 0., 0. 0.
0.0434 0. 0, 0, 0.
0 0 0 0 0.434...]]

27/4/22
4:30pm.

```
# f_imp2 = Pd.DataFrame (index = X.columns, data = final model
        .feature_importances_, columns = ["importance-
        Feature"])
```

# f_imp2

Out:

| | importance_Feature |
|---|---|
| cap_shape_c | 0.0 |
| cap-shape-f | 0.0 |
| ⋮ | |
| habitat-p | 0.0 |
| habitat-u | 0.0 |

# important = f_imp2 [ f_imp2 [ "importance_feature" ] >0 ]

# important . Sort_Values ( " importance-feature")

Out:

| | importance_Feature |
|---|---|
| cap_color_w | 0.043478 |
| odor_c | 0.043478 |
| odor_f | 0.043478 |
| odor_p | 0.043478 |
| ⋮ | |
| order_n | 0.130435 |
| gill_size_n | 0.173913 |

# plt. figure [fig_size = (14,6), dpi=200)

# sns. barplot (data = important. Sort_values ("importance_feature"), x = important. index, y = "importance_Feature")

# plt.xticks (rotation = 90)

# plt. show ()