In [1]:
```python
import numpy as np
```

In [ ]:
```python
##1. fit function will create dictionaries and then we write the predict function which takes the dictionary and
## takes the testing data and tells us predictions corrosponding to that testing data.

def fit(X_train, Y_train):
    ##2. result will be a dictionary. This result will have some keys.
    result = {}


    ##2. result will have keys as the distinct(unique) values of Y_train.
    class_values = set(Y_train)

    ##3. Now we have our top level keys.
    ##3. We are going through all possible classes.
    for current_class in class_values:
        ##4. result of current_class will again be a dictionary.
        ##4. and this dictionary will again have all possible features.
        result[current_class] = {}

        ## 16. One more thing we will be storing that is -
        result['total_data'] = len(Y_train)

        ## 11. What we do here is we find out just the training data which has the class as current_class.
        current_class_rows = (Y_train == current_class)
        ## 11. above will return True or false array.

        ## 12. To get current training data, we need to do is
        X_train_current = X_train[current_class_rows]
        ## 12. We will get only those rows which has current_class_rows == True.

        ## 13. Similarly,
        Y_train_current = Y_train[current_class_rows]


        ##5. to find all possible features we do this.
        num_features = X_train.shape[1]


        ## 15. Aling with this, we will store total count of training data which belongs to current class.
        result[current_class]['total_count'] = len(Y_train_current)

        ##6. lets pass thorugh each feature.
        for j in range(1, num_features + 1):   ## so that we get count of features from 1.
            ## 34. We changed the range from 0 - n to 1 - n+ 1, which led to a problem that we are trying to acce
            ## 34. But when we try to access in X_train, we need to go till n - 1 due to indexing.

            ##7. so for each of these features we will be creating a new dictionary.
            result[current_class][j] = {}

            ##8. what we want to store in this dict is lets say [a1][2][] for class a1, for feature 2, what all
            ##8. possible value that feature 2 can take, and for each of this value, we want to store its count.
            ##8. to find,

            ## 34. j - 1 is due to because of above reasons.
            all_possible_values = set(X_train[:, j - 1])

            ##9. then we go through all possible value in the set.
            for current_value in all_possible_values:

                ##10. now for each value, we want to find the count.
                ##10. count of for all training data points where the y is current class.
                ##10. In how many of them do you have jth feature values as current feature value.
                ##10. In a nutshell, we want to look at training data which has class as current_class.


                ## 14. Now out of those current values, we will take only those which have jth feature = current_
                ## 14. X_train_current[:, j] == current_value will give me a True - false numpy array.
                ## 14. (X_train_current[:, j] == current_value).sum() will give us whereever it has True, it will
                ## 14. and for false, it will count as 0.so it becomes count value of 1.

                ## 34. j-1 is due to above reasons.
                result[current_class][j][current_value] = (X_train_current[:, j - 1] == current_value).sum()


    return result
```

In [ ]:
```python
def predict(dictionary, X_test):
    ##17. Now for testing data, we are supposed to predict the classes and return that as another array.
    y_pred = []
    for x in X_test:

        ##18. we will go through the testing data, lets say we will have another function which will give us x_cl
        x_class = predictSinglePoint(dictionary, x)
```

```python
        ## 19. we will append x_class to the y_pred.
        y_pred.append(x_class)

    ##19. we will return y_pred
    return y_pred
```

In [ ]:
```python
def predictSinglePoint(dictionary, x):
    ## 20. Now for single point we need to predict which class it belongs to.
    ## 20. what we need to do is we need to go through all possible classes and for each class we calculate
    ## 20. the formula we defined.

    ## 21. to find all possible classes, we defined the keys as classes so we do this.
    classes = dictionary.keys()

    ## 23. while doing all of this, we can maintain best probablity and store the best class as well
    best_p = -1000
    best_class = -1

    ## Just a naive concept
    first_run = True


    for current_class in classes:

        ## 34. There is also another class we added that is count of total data. SO we dont have to include it.
        if current_class == 'total_data':
            continue

        ## 22. we will go thoriugh each class and for each clas we find the the probablity of each class.
        p_current_class = probablity(dictionary, x, current_class)
        if (first_run or p_current_class > best_class):   ## First_run just so to be sure that it updates the val
            best_p = p_current_class
            best_class = current_class
        first_run = False
    return best_class
```

In [ ]:
```python
def probablity(dictionary, x , current_class):

    ## 24. It has two components, y = ai(current_class)
    ## 25. To find the probablity, we need to find number of class value and total class number.
    ## 25. to do so, we do the following

    ## 35. We are changing it to a log probablity function
    ##output = dictionary[current_class]['total_count']/dictionary['total_data']
    output = np.log(dictionary[current_class]['total_count']) - np.log(dictionary['total_data'])


    ## 26. what we  will do now is we will find it for each individual feature and what we will do is we will kee
    ## 26. it with output.

    ## 27. what we will do now is we will go through each possible features.
    ## 27. to find no of features, we have keys in classes dictionary of features.
    num_features = len(dictionary[current_class].keys()) - 1  ## -1 is because we have one extra key i.e. total_c

    ## 28. calculating feature number here.
    for j in range(1, num_features+ 1):

        ## 29. now we wwant to find p(X^j = x^j/ y = ai).

        ## 30. this is for selecting current j.

        ## 34. Similarly here we need to access j -1 due to indexing.
        xj = x[j]

        ## 30. Numerator.
        ## 32. Applying laplace correction, we need to add 1 in numerator
        count_current_class_with_value_xj = dictionary[current_class][j][xj]  + 1

        ## 30. now finding denominator.
        ## 33. Applying laplace correction, we need to find how many values this feature j can take.
        count_current_class = dictionary[current_class]['total_class'] + len(dictionary[current_class][j].keys())

        ## 35. Changing it to log probablity
        ##current_xj_probablity = count_current_class_with_value_xj/count_current_class
        current_xj_probablity = np.log(count_current_class_with_value_xj) - np.log(count_current_class)


        ## 35. changing it to log probablity
        ##output = output*current_xj_probablity
        output = output + current_xj_probablity

    return output

## 31. To apply laplace correction, we need to change numerator as well as denominator.
```

In [ ]:
```
## for the running part, refer naivebayes3.
```