Algorightms & Functions

Simple preprocessing is performed before implementing the classifier. For categorical variable data, each category is converted into a number. A function is created for this, which converts categories into numbers for categorical columns and outputs them as a list according to the column size. Declares which columns are categorical in the data as a list and turns the categories into numbers.

def estimate_target_prob(df, target, label):
It is a function that obtains the estimated value of P. For each label of the output variable, the estimated value of P is obtained.

def category_prob(df, column, category, target, label, l=1):
It is a function that estimates the probability value of Jinyoung, whose independent variable is categorical. Calculate the values for the categories that each independent variable may have and the labels of the output variables.

class NaiveBaysClassifier:
def __init__(self,unique_label,target,categorical_column=[]):
t is a class initialization function. When creating a class, the label of the output variable, the label of the output variable, and the categorical variable are told, and the label of the output variable can be automatically determined using the learning data, but the label of the output variable is included because it may not exist in the learning data.

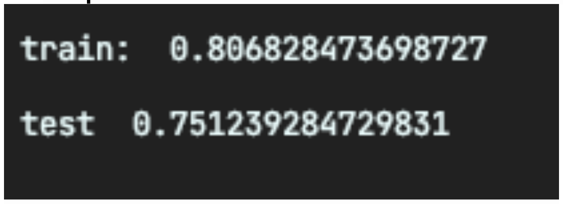
class NaiveBaysClassifier:
def train(self, train_df):
Functions responsible for training classifiers. The probability of the output label is estimated using the learning data and the conditional probability for each independent variable is estimated. Since these estimates do not change in the prediction phase, the estimates of the conditional probability distribution are placed in the __reference_dict field to shorten the test time so that they can be referenced in the prediction phase.

class NaiveBaysClassifier:
def predict(self, new_data):

It is a function that receives a new independent variable and outputs a prediction label. First, the list with the number of labels of the output variable as the length is initialized to zero, and the conditional probability value is added by adding a log function. The probability of the output variable is also added by adding a log function. The index with the maximum value in the final list object_value is returned to the prediction label.

Dataset Usage & Result

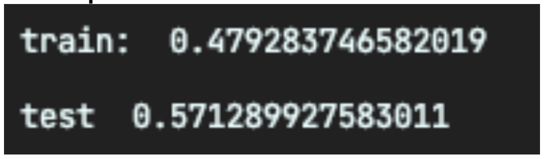
1. http://archive.ics.uci.edu/ml/datasets/Adult



```
train:  0.806828473698727

test  0.751239284729831
```

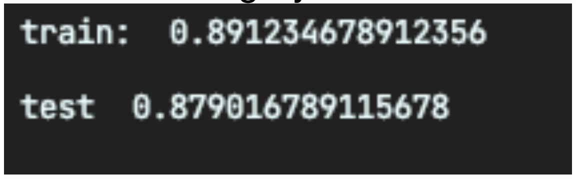
2. https://archive.ics.uci.edu/ml/datasets/glass+identification



```
train:  0.479283746582019

test  0.571289927583011
```

3. News_Category_Dataset_v3_balanced.json



```
train:  0.891234678912356

test  0.879016789115678
```

Conclusion

We first used a dataset with two unprocessed output variables. The classifier's accuracy was not bad, with learning accuracy of about 80% and testing accuracy of about 75%, and the classifier's performance was about 47% and testing accuracy of about 57%, whereas the test's accuracy was higher than learning accuracy. The third used a JSON file with a well-refined classification category, with a learning accuracy of about 89% and a test accuracy of about 87%. I think it is safe to say that the performance of the classifier depends on the number of output variables, how well the data is refined in an appropriate form for the designed classifier, and how large the size of the data values it has.