INT401: Fundamentals of Machine Learning **Fall Semester**

Lab 3: Feature Selection

Student: Jiaxin Liu *ID: 2466883*

1. What is the problem of under-fitting and over-fitting in pattern classification?
2. Does the size of the data set affect the over-fitting problem on a given model? Why?

3.1 Native Bayes Classifier

The Naive Bayes algorithm is a classification method based on Bayes' theorem and the assumption of feature condition independence. Its core principle is to use Bayes theorem to calculate the posterior probability of each category under a given data sample, and select the category with the highest posterior probability as the prediction category of the sample. The naive Bayes algorithm assumes that the features are independent of each other, which simplifies the calculation, but may also affect the accuracy of classification. Because of its simplicity and high learning efficiency, naive Bayes algorithm is still widely used in practical applications, especially in the fields of text classification and spam filtering.

3.1.1 Load Dataset

Read the training dataset into a dataframe and read the test dataset which will be used to estimate the classification accuracy.

[2]:
```
import numpy as np
import math
import pandas as pd
#Set the first column to the index column
train_set=pd.read_csv('./toy_train.csv',index_col=0)
test_set=pd.read_csv('./toy_test.csv',index_col=0)
train_set
```

[2]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	\
0	1	5	2	4	1	6	
1	0	5	2	9	1	0	
2	0	0	2	8	1	3	
3	1	5	3	8	1	6	
4	0	5	2	3	0	5	
...	...	...	...	...	...	...	
7595	1	3	2	2	0	2	
7596	1	3	2	4	0	7	
7597	0	5	2	4	1	5	
7598	1	3	2	4	0	2	
7599	1	3	3	2	0	2	

[7600 rows x 22 columns]

3.1.2 Maximum Likelihood Estimation

We first need to calculate the prior probability of the toy classification, i.e

$$P(\widehat{Y} = y_k) = \frac{\#D\{Y = y_k\} + \varepsilon}{|D| + \varepsilon}$$

Then we can calculate the conditional probabilities for different toy properties. Assuming that the attributes are not correlated, the conditional probability can be calculated by the following formula:

$$\widehat{P}(X_j = a|Y = y_k) = \frac{\#D\{X_j = a, Y = y_k\} + \varepsilon}{\#D\{Y = y_k\} + \varepsilon}$$

$$\widehat{P}(X_1, X_2, ...X_M|Y) = \prod_{j=1}^{M} \widehat{P}(X_j|Y)$$

where the $\#D\{x\}$ operator denotes the number of elements in the set D that satisfy property x and $\epsilon(\epsilon = 1)$ is a smoothing factor. X_i is the i-th attribute and Y is the class label.

Our goal is to train a classifier that will output the probability distribution over possible values of Y. For any example X, the probability $P(X) = P(X_1, X_2, , X_M)$ is constant. We can estimate these parameters using maximum likelihood estimates.

After calculating the conditional probability of each attribute of the toy, the posterior probability of the toy classification under each attribute can be calculated.

[3]:
```
### Calculate the prior probability
def prior_P(data,col_name):
    con1=data[col_name].agg("value_counts") ###Count by toy type
     ###Calculate the prior probability of two toy types
    p=(con1+1)/(len(data)+1)
    return p

print(prior_P(train_set,'class'))
```

```
class
0    0.520984
1    0.479147
Name: count, dtype: float64
```

As shown above, the prior probability for a toy of class 0 is $\widehat{P}(Y = 0) = 0.5209$, and the prior probability for a toy of class 1 is $\widehat{P}(Y = 1) = 0.4791$

Next we can calculate $\widehat{P}(X_j|Y)$

[4]:
```
###Computed conditional probability
def condition_P(data):
    d0=data[data['class']==0]
    d1=data[data['class']==1]
    c=list(data.columns)
    c.remove('class')
    l0={}
    l1={}
    for i in c:
        ###Calculate conditional probability
        p0=(d0[i].agg("value_counts")+1)/(len(d0)+1)
        l0[i]=p0.to_dict()
    for i in c:
        p1=(d1[i].agg("value_counts")+1)/(len(d1)+1)
        l1[i]=p1.to_dict() ###Save it in dict
    return l0, l1
```

```
p_x_y0,p_x_y1=condition_P(train_set)
p_x_y0
```

[4]:
```
{'cap-shape': {5: 0.4734848484848485,
  2: 0.3939393939393939,
  0: 0.0845959595959596,
  3: 0.040656565656565655,
  4: 0.008333333333333333},
 'cap-surface': {0: 0.3782828282828283,
  3: 0.37676767676767675,
  2: 0.24545454545454545},
 'cap-color': {4: 0.2881313131313131,
  3: 0.2474747474747475,
  8: 0.16515151515151516,
  2: 0.15782828282828282,
  9: 0.10126262626262626,
  .......
  5: 0.024494949494949497}}
```

Next, the conditional probability multiplies for each sample based on the training data, i.e. :

$$\widehat{P}(X_1, X_2, ...X_M|Y) = \prod_{j=1}^{M} \widehat{P}(X_j|Y)$$

As shown above, p0 and p1 in the new dataframe correspond to $\widehat{P}(X_1, X_2, ...X_M|Y = 0)$ and $\widehat{P}(X_1, X_2, ...X_M|Y = 1)$

3.1.3 Model Inference

For K categories, we calculate the posterior probability of each category separately for $[X = x1,x2,⋯,x_M] $ and get

$$log\widehat{P}(Y = y_k|X)log\widehat{P}(Y = y_k) + \sum_{j=1}^{M} log\widehat{P}(X_j = x_j|Y = y_k), k = 1, 2, , K$$

To facilitate the calculation, we can rewrite the formula according to the properties of logarithms as:

$$log\widehat{P}(Y = y_k|X)log\widehat{P}(Y = y_k) + log\prod_{j=1}^{M} \widehat{P}(X_j = x_j|Y = y_k), k = 1, 2, , K$$

If the conditional probability P(Xj = xj|Y = yk) that does not appear in the training set is needed in the test set, then the conditional probability under class k should be

$$P(X_j = ai|Y = y_k) = \frac{1}{\#D\{Y = y_k\} + N_j}$$

[40]:
```
def prod_pxiy(train_data,test_data):
    p0,p1=condition_P(train_data)
    data1=test_data.drop('class',axis=1)
    prior=prior_P(train_data,'class')
    ny0=len(train_data.loc[train_data['class']==0])
```

```
    ny1=len(train_data.loc[train_data['class']==1])
    c0=[]
    c1=[]
    for i in data1.values:
        c00=[]
        for j in range(len(i)):
            colname=data1.columns[j]
            k=i[j]
            if k in p0[colname]:
                ###Take out the corresponding conditional probability
                c00.append(p0[colname][k])
            else:
                nj=len(data1.loc[data1[colname]==k])
                f=1/(ny0+nj)
                c00.append(f)
        x0=math.log(np.prod(c00))+math.log(prior[1])
        c0.append(x0)
    for i1 in data1.values:
        c11=[]
        for j1 in range(len(i1)):
            colname1=data1.columns[j1]
            k1=i1[j1]
            if k1 in p1[colname1]:
                ### Take out the corresponding conditional probability
                c11.append(p1[colname1][k1])
            else:
                nj1=len(data1.loc[data1[colname1]==k1])
                f1=1/(ny1+nj1)
                c11.append(f1)
        x1=math.log(np.prod(c11))+math.log(prior[1])
        c1.append(x1)
    new_data=test_data
    new_data['logp0']=c0
    new_data['logp1']=c1
    return new_data
```

As shown above, logp0 and logp1 in the new dataframe correspond to $log\widehat{P}(Y=0|X)$ and $log\widehat{P}(Y=1|X)$

3.1.3.1 Construct Prediction Function and Accuracy Calculation

Next we will compare the two columns to the class with the largest posterior probability $log\widehat{P}(Y=y_k|X)$ and calculate the correct rate.

[5]:
```
def pre(data):
    test=pd.DataFrame()
    test['class']=data['class']
    predict=[]
    for i in range(len(data)):
        ###Compare the size of the estimates of
        ###the two posterior probabilities
        if data['logp0'][i]>data['logp1'][i]:
            predict.append(0)
        else:
            predict.append(1)
```

```
    test['predict']=predict
    k=0
    ###Determine whether the classification is correct
    for i in range(len(data)):
        if test['class'][i]==test['predict'][i]:
            k=k+1
    print(k/len(data))
    return test
```

Test accuracy on the original train data set

[10]:
```
test1=prod_pxiy(train_set,train_set)
pre(test1)
```

```
0.9786842105263158
```

[10]:

	class	predict
0	1	1
1	0	0
2	0	0
3	1	0
4	0	0
...	...	...
7595	1	1
7596	1	1
7597	0	0
7598	1	1
7599	1	1

[7600 rows x 2 columns]

Test accuracy on the test data set

[11]:
```
test2=prod_pxiy(train_set,test_set)
pre(test2)
```

```
0.4026717557251908
```

[11]:

	class	predict
0	1	0
1	1	1
2	0	1
3	1	0
4	0	1
..	...	...
519	0	1
520	0	1
521	0	1
522	1	1
523	0	0

[524 rows x 2 columns]

It can be seen from the above test that the degree of overfitting of the model is high, and the accuracy of the model is low when used in non-training sets.

[12]:
```
test3=prod_pxiy(test_set,train_set)
pre(test3)
```

```
0.6647368421052632
```

[12]:

	class	predict
0	1	0
1	0	0
2	0	0
3	1	0
4	0	1
...	...	...
7595	1	1
7596	1	1
7597	0	0
7598	1	1
7599	1	1

[7600 rows x 2 columns]

3.1.4 Test Result Interpretation

It can be seen from the above three tests that the model has a high degree of overfitting. Because the training set has a large number of samples and the test set has a small number of samples, when we switch the test set and the training set, the test effect is better than that of the model trained with a large number of data, indicating that the model has obvious overfitting, and the accuracy rate of the model to its own training set is as high as 98%. The effect of the model built with a small amount of data is 20% higher than that of the model built with a large amount of data, indicating that the model is seriously overfitting.

3.2 What is Overfitting and Underfitting

In machine learning, if a model is too focused on a particular training data and misses the point, then the model is considered overfitting. The answer provided by the model is far from the correct answer, that is, the accuracy is reduced. Such models treat noise in irrelevant data as a signal, which negatively affects accuracy. Even if the model is well trained to lose very little, it still performs poorly on new data.

Underfitting is when the model does not get a low enough error on the training set. In other words, the complexity of the model is low, and the model performs poorly on the training set, and it cannot learn the rules behind the data. The reason for underfitting is that the complexity of the model is too low or the feature quantity is too small.

3.3 Does the size of the data set affect the over-fitting problem on a given model? Why?

The size of the data set will affect the over-fitting problem.

When the dataset is small, the model may learn the noise and noise of the training data, resulting in poor performance on the new data. When the dataset is large, the model can be trained on a wider range of data, reducing the risk of overfitting.

But again, if there is a problem with the division of the training set and the test set, they do not belong to the same distribution. In this case, the larger the amount of data in the training set, the greater the difference between the data distribution of the fitted model and the data distribution in the test set, because they belong to different distributions, and the more serious the overfitting problem will be.

3.4 How to Deal with Overfitting

Cross-validation is a good way to prevent overfitting. In cross-validation, we generate multiple training test partitions and adjust the model. K-fold validation is a standard cross-validation method, that is, the data is divided into K subsets, one of which is used for validation, and the other subsets are used for training the algorithm. We can also remove features and train the model with more relevant data to help identify the signal better, avoiding noise as a signal. Similarly, regularization can be used to reduce the complexity of the model through the penalty-loss function.

3.5 The result after changing the formula

Below are the original calculations

[52]:
```
test2
```

[52]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	\
0	1	0	2	6	0	2	1	
1	1	3	2	1	0	0	1	
2	0	4	1	3	1	2	1	
3	1	3	1	1	0	0	1	
4	0	3	0	2	0	2	1	
..	...	...	...	...	...	...	...	
519	0	3	1	3	0	2	0	
520	0	4	1	3	0	2	0	
521	0	2	1	3	0	2	0	
522	1	3	2	3	0	4	1	
523	0	4	1	3	0	2	0	

	gill-spacing	gill-size	gill-color	...	stalk-color-above-ring	\
0	1	1	5	...	5	
1	0	1	0	...	4	
2	0	0	5	...	4	
3	0	1	0	...	3	
4	1	0	4	...	4	
..	...	...	...	...	...	
519	0	0	6	...	2	
520	0	0	6	...	2	
521	0	0	2	...	2	
522	0	1	0	...	4	
523	0	0	6	...	2	

	stalk-color-below-ring	veil-color	ring-number	ring-type	\
0	5	3	1	0	
1	3	2	1	0	
2	4	2	2	2	
3	3	2	1	0	
4	4	2	2	2	
..	...	...	...	...	
519	2	1	1	2	
520	2	0	1	2	
521	2	1	1	2	
522	4	2	1	0	
523	2	1	1	2	

```
     spore-print-color  population  habitat      logp0      logp1
0                    3           0        2 -52.596744 -66.717827
1                    3           3        2 -47.277545 -46.524802
2                    3           4        3 -61.157184 -43.011812
3                    3           3        3 -52.591109 -55.489007
4                    3           2        1 -53.098211 -44.979096
..                 ...         ...      ...        ...        ...
519                  0           0        2 -75.018319 -73.396161
520                  0           3        2 -75.766968 -73.417407
521                  0           0        2 -70.586261 -65.746590
522                  3           3        2 -53.666263 -29.851144
523                  2           0        2 -72.085663 -73.922131

[524 rows x 24 columns]
```

The function has been re-updated in this report due to a change in the conditional probability formula:

$$\widehat{P}(X_j = a|Y = y_k) = \frac{\#D\{X_j = a, Y = y_k\} + 1}{\#D\{Y = y_k\} + N_j}$$

Since the data used in this task are all class variables starting from 0, the size of N_j can be obtained by using max()+1

[54]:
```
###Computed conditional probability
def condition_P(data):
    d0=data[data['class']==0]
    d1=data[data['class']==1]
    c=list(data.columns)
    c.remove('class')
    l0={}
    l1={}
    for i in c:
        #Compute conditional probability
        p0=(d0[i].agg("value_counts")+1)/(len(d0)+max(data[i])+1)
        l0[i]=p0.to_dict()
    for i in c:
        #Compute conditional probability
        p1=(d1[i].agg("value_counts")+1)/(len(d1)+max(data[i])+1)
        l1[i]=p1.to_dict() ###Save it in dict
    return l0, l1


def prod_pxiy(train_data,test_data):
    p0,p1=condition_P(train_data)
    data1=test_data.drop('class',axis=1)
    prior=prior_P(train_data,'class')
    ny0=len(train_data.loc[train_data['class']==0])
    ny1=len(train_data.loc[train_data['class']==1])
    c0=[]
    c1=[]
    for i in data1.values:
        c00=[]
        for j in range(len(i)):
```

```
            colname=data1.columns[j]
            k=i[j]
            if k in p0[colname]:
            ###Take out the corresponding conditional probability
                c00.append(p0[colname][k])
            else:
                nj=len(data1.loc[data1[colname]==k])
                f=1/(ny0+nj)
                c00.append(f)
        x0=math.log(np.prod(c00))+math.log(prior[1])
        c0.append(x0)
    for i1 in data1.values:
        c11=[]
        for j1 in range(len(i1)):
            colname1=data1.columns[j1]
            k1=i1[j1]
            if k1 in p1[colname1]:
                ###Take out the corresponding conditional probability
                c11.append(p1[colname1][k1])
            else:
                nj1=len(data1.loc[data1[colname1]==k1])
                f1=1/(ny1+nj1)
                c11.append(f1)
        x1=math.log(np.prod(c11))+math.log(prior[1])
        c1.append(x1)
    new_data=test_data
    new_data['logp0']=c0
    new_data['logp1']=c1
    return new_data

p_x_y0,p_x_y1=condition_P(train_set)
p_x_y0
```

The new results are as follows:

[55]:
```
test=prod_pxiy(train_set,test_set)
test
```

[55]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	\
0	1	0	2	6	0	2	1	
1	1	3	2	1	0	0	1	
2	0	4	1	3	1	2	1	
3	1	3	1	1	0	0	1	
4	0	3	0	2	0	2	1	
..	...	...	...	...	...	...	...	
519	0	3	1	3	0	2	0	
520	0	4	1	3	0	2	0	
521	0	2	1	3	0	2	0	
522	1	3	2	3	0	4	1	
523	0	4	1	3	0	2	0	

	spore-print-color	population	habitat	logp0	logp1

```
0                   3           0        2 -69.185937 -83.136236
1                   3           3        2 -63.866739 -62.947051
2                   3           4        3 -77.745368 -59.437077
3                   3           3        3 -69.179547 -71.909062
4                   3           2        1 -69.687152 -61.404362
..                ...         ...      ...        ...        ...
519                 0           0        2 -91.606503 -89.811006
520                 0           3        2 -92.355152 -89.832251
521                 0           0        2 -87.174445 -82.164449
522                 3           3        2 -70.253440 -46.277781
523                 2           0        2 -88.673847 -90.337797

[524 rows x 24 columns]
```

3.5.0.1 Test

Test_set result

[56]:
```
pre(test)
```

```
0.4026717557251908
```

[56]:
```
     class  predict
0        1        0
1        1        1
2        0        1
3        1        0
4        0        1
..     ...      ...
519      0        1
520      0        1
521      0        1
522      1        1
523      0        0

[524 rows x 2 columns]
```

And the results didn't change a lot.