1、 P(AB) 表示的是事件 A 与事件 B 同时发生的概率， P(A|B) 表示的是事件 B 已经发生的条件下，事件 A 发生的概率。

- [x] A、对
- [ ] B、错

2、 从 1,2,...,15 中小明和小红两人各任取一个数字，现已知小明取到的数字是 5 的倍数，请问小明取到的数大于小红取到的数的概率是多少?

- [ ] A、7/14
- [ ] B、8/14
- [x] C、9/14
- [ ] D、10/14

1、 对以往数据分析结果表明，当机器调整得良好时，产品的合格率为 98% ，而当机器发生某种故障时，产品的合格率为 55% 。每天早上机器开动时，机器调整得良好的概率为 95% 。计算已知某日早上第一件产品是合格时，机器调整得良好的概率是多少?

- [ ] A、0.94
- [ ] B、0.95
- [ ] C、0.96
- [x] D、0.97

2、 一批产品共 8 件，其中正品 6 件，次品 2 件。现不放回地从中取产品两次，每次一件，求第二次取得正品的概率。

- [ ] A、1/4
- [ ] B、1/2
- [x] C、3/4
- [ ] D、1

```
import numpy as np


class NaiveBayesClassifier(object):
    def __init__(self):

        self.label_prob = {}

        self.condition_prob = {}
    def fit(self, feature, label):
        '''
        对模型进行训练，需要将各种概率分别保存在self.label_prob和
self.condition_prob中
        :param feature: 训练数据集所有特征组成的ndarray
        :param label:训练数据集中所有标签组成的ndarray
        :return: 无返回
        '''
```

```
        #********** Begin **********#
        row_num=len(feature)
        col_num=len(feature[0])
        for c in label:
            if c in self.label_prob:
                self.label_prob[c]+=1
            else:
                self.label_prob[c]=1;
        for key in  self.label_prob.keys():
            self.label_prob[key]/=row_num
            self.condition_prob[key]={}
            for i in range(col_num):
                self.condition_prob[key][i]={}
                for k in np.unique(feature[:,i],axis=0):
                    self.condition_prob[key][i][k]=0
        for i in range(len(feature)):
            for j in range(len(feature[i])):
                if feature[i][j] in self.condition_prob[label[i]]:
                    self.condition_prob[label[i]][j][feature[i]
[j]]+=1
                else:
                    self.condition_prob[label[i]][j][feature[i]
[j]]=1
        for label_key in self.condition_prob.keys():
            for k in self.condition_prob[label_key].keys():
                total=0
                for v in self.condition_prob[label_key]
[k].values():
                    total +=v
                for kk in self.condition_prob[label_key][k].keys():
                    self.condition_prob[label_key][k][kk] /=total

        #********** End **********#

    def predict(self, feature):
        '''
        对数据进行预测，返回预测结果
        :param feature:测试数据集所有特征组成的ndarray
        :return:
        '''
        # ********** Begin **********#
```

```
        result =[]
        for i,f in enumerate(feature):
            prob=np.zeros(len(self.label_prob.keys()))
            ii=0
            for label,label_prob in self.label_prob.items():
                prob[ii]=label_prob
                for  j  in range(len(feature[0])):
                    prob[ii] *= self.condition_prob[label][j][f[j]]
                ii+=1
            result.append(list(self.label_prob.keys())
[np.argmax(prob)])
        return np.array(result)
        #********* End *********#
```

```
import numpy as np

class NaiveBayesClassifier(object):
    def __init__(self):

        self.label_prob = {}

        self.condition_prob = {}

    def fit(self, feature, label):
        '''
        对模型进行训练，需要将各种概率分别保存在self.label_prob和
self.condition_prob中
        :param feature: 训练数据集所有特征组成的ndarray
        :param label:训练数据集中所有标签组成的ndarray
        :return: 无返回
        '''

        #********* Begin *********#
        row_num=len(feature)
        col_num=len(feature[0])
        unique_label_count=len(set(label))
        for c in label:
            if c in self.label_prob:
                self.label_prob[c]+=1
```

```
            else:
                self.label_prob[c]=1
        for key in self.label_prob.keys():
            self.label_prob[key]+=1
            self.label_prob[key]/=(unique_label_count+row_num)
            self.condition_prob[key]={}
            for i in range(col_num):
                self.condition_prob[key][i]={}
                for k in np.unique(feature[:,i],axis=0):
                    self.condition_prob[key][i][k]=1
        for i in range(len(feature)):
            for j in range(len(feature[i])):
                if feature[i][j] in self.condition_prob[label[i]]:
                    self.condition_prob[label[i]][j][feature[i]
[j]]+=1

        for label_key in self.condition_prob.keys():
            for k in self.condition_prob[label_key].keys():
                total =len(self.condition_prob[label_key].keys())
                for v in self.condition_prob[label_key]
[k].values():
                    total+=v
                for kk in self.condition_prob[label_key][k].keys():
                    self.condition_prob[label_key][k][kk]/=total
                #********** End **********#


    def predict(self, feature):
        '''
        对数据进行预测，返回预测结果
        :param feature:测试数据集所有特征组成的ndarray
        :return:
        '''

        result = []
        # 对每条测试数据都进行预测
        for i, f in enumerate(feature):
            # 可能的类别的概率
            prob = np.zeros(len(self.label_prob.keys()))
            ii = 0
            for label, label_prob in self.label_prob.items():
                # 计算概率
```

```
                prob[ii] = label_prob
                for j in range(len(feature[0])):
                    prob[ii] *= self.condition_prob[label][j][f[j]]
                ii += 1
            # 取概率最大的类别作为结果
            result.append(list(self.label_prob.keys())
[np.argmax(prob)])
        return np.array(result)
```

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfTransformer

def news_predict(train_sample, train_label, test_sample):
    '''
    训练模型并进行预测，返回预测结果
    :param train_sample:原始训练集中的新闻文本，类型为ndarray
    :param train_label:训练集中新闻文本对应的主题标签，类型为ndarray
    :param test_sample:原始测试集中的新闻文本，类型为ndarray
    :return 预测结果，类型为ndarray
    '''
    #********* Begin *********#
    vec=CountVectorizer()
    train_sample=vec.fit_transform(train_sample)
    test_sample=vec.transform(test_sample)

    tfidf=TfidfTransformer()

    train_sample =tfidf.fit_transform(train_sample)
    test_sample=tfidf.transform(test_sample)
    mnb=MultinomialNB(alpha=0.01)
    mnb.fit(train_sample,train_label)
    predict=mnb.predict(test_sample)
    return predict
    #********* End *********#
```