HW1-BAYES

SA22011090 余致远

朴素贝叶斯分类器（Naive Bayes Classifier）

数据集：Bayesian_Dataset_train.csv, Bayesian_Dataset_test.csv。

数据描述：列名分别为"年纪、工作性质、家庭收入、学位、工作类型、婚姻状况、族裔、性别、工作地点"，最后一列是标签，即收入是否大于 50k 每年。

任务描述：使用朴素贝叶斯（Naïve Bayesian）预测一个人的收入是否高于 50K 每年。

要求输出：

1）结果统计，例如 precision、recall、F1 score 等；

2）csv 文件，在 test 文件最后增加一列，填入模型预测的收入标签（<=50K 或>50K）

Optional：探索不同参数对结果的影响。

In [264…

```
import pandas as pd

train_file_path = 'datasets/Bayesian_Dataset_train.csv'
test_file_path  = 'datasets/Bayesian_Dataset_test.csv'
train_df = pd.read_csv(train_file_path, names=['age','job_nature','family_income','d
test_df = pd.read_csv(test_file_path, names=['age','job_nature','family_income','deg
# train_df

train_df
```

Out[264]:

	age	job_nature	family_income	degree	marriage	job_type	in_family	race	gende
0	39	State-gov	77516	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Ma
1	50	Self-emp-not-inc	83311	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Ma
2	38	Private	215646	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Ma
3	53	Private	234721	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Ma
4	28	Private	338409	Bachelors	Married-civ-spouse	Prof-specialty	Wife	Black	Fema
...	...	...	...	...	...	...	...	...	
27197	27	Private	257302	Assoc-acdm	Married-civ-spouse	Tech-support	Wife	White	Fema
27198	40	Private	154374	HS-grad	Married-civ-spouse	Machine-op-inspct	Husband	White	Ma
27199	58	Private	151910	HS-grad	Widowed	Adm-clerical	Unmarried	White	Fema
27200	22	Private	201490	HS-grad	Never-married	Adm-clerical	Own-child	White	Ma
27201	52	Self-emp-inc	287927	HS-grad	Married-civ-spouse	Exec-managerial	Wife	White	Fema

27202 rows × 11 columns

In [265…

```
# jobNatureMap = {elem:index+1 for index, elem in enumerate(set(train_df['job_nature
# train_df['job_nature'] = train_df['job_nature'].map(jobNatureMap)

# pd.factorize() is a better way

map_dict = {}
for column in train_df.columns:
    if column == 'age' or column == 'family_income':
        continue
    factorized_column, map = train_df[column].factorize()
    train_df[column] = factorized_column
    map_dict[column] = {map[i]:i for i in range(len(map))}

# train_df
# factorized_column
map_dict['race']
```

Out[265]:

```
{' White': 0,
 ' Black': 1,
 ' Asian-Pac-Islander': 2,
 ' Amer-Indian-Eskimo': 3,
 ' Other': 4}
```

In [266…
```
# map_dict['race'][[1,2,1]]
```

In [267…
```
for column in test_df.columns:
    if column == 'age' or column == 'family_income':
        continue
    test_df[column] = test_df[column].replace(map_dict[column])

test_df
```

Out[267]:

	age	job_nature	family_income	degree	marriage	job_type	in_family	race	gender	wor
0	52	1	209642	1	1	1	1	0	0	
1	59	2	109015	1	2	10	4	0	1	
2	56	5	216851	0	1	10	1	0	0	
3	23	5	190709	6	0	11	0	0	0	
4	31	2	507875	4	1	8	1	0	0	
...	...	...	...	...	...	...	...	...	...	
2955	24	2	381895	2	2	8	4	0	1	
2956	18	2	436163	2	0	3	3	0	0	
2957	25	5	514716	0	0	0	3	1	1	
2958	46	2	42972	3	1	3	2	0	1	
2959	30	2	77266	1	2	6	0	0	0	

2960 rows × 11 columns

In [268…
```
train_features, train_labels = train_df.iloc[:,:-1], train_df.iloc[:,-1]
test_features, test_labels = test_df.iloc[:,:-1], test_df.iloc[:,-1]

# test_features
```

In [269…
```
from sklearn.naive_bayes import CategoricalNB
from sklearn.metrics import precision_recall_fscore_support, accuracy_score

model = CategoricalNB()
model.fit(train_features, train_labels)

y_pred = model.predict(test_features)
y_pred
```

Out[269]:
```
array([1, 0, 1, ..., 0, 1, 0], dtype=int64)
```

In [270…
```
precision, recall, F1_score, _ = precision_recall_fscore_support(test_labels, y_pred
acc = accuracy_score(test_labels, y_pred)
print("precision: {} \nrecall: {} \nF1 score: {} \naccuracy: {}".format(precision, r
```

```
precision: [0.90317776 0.5845666 ]
recall: [0.82233273 0.73930481]
F1 score: [0.86086133 0.65289256]
accuracy: 0.8013513513513514
```

In [271…
```
# mostly borrowed from https://juejin.cn/post/6865193399784472583
import numpy as np
import math
```

```
from scipy.stats import multivariate_normal

class NaiveBayes:
    def __init__(self):
        self.model = None

    # process X_train
    def summarize(self, X):
        # summaries = [(self.mean(i), self.stdev(i)) for i in zip(*X)]
        summaries = [(np.mean(i), np.var(i)) for i in zip(*X)]
        return summaries

    def fit(self, X, y):
        labels = list(set(y))
        data = {label: [] for label in labels}
        for feature, label in zip(X, y):
            data[label].append(feature)
        self.model = {label: self.summarize(feature) for label, feature in data.ite
        return self.model

    # 计算概率
    def calculate_probabilities(self, X):
        probabilities = {}
        for label, feature in self.model.items():
            probabilities[label] = 1
            for i in range(len(feature)):
                mean, stdev = feature[i]
                probabilities[label] *= multivariate_normal.pdf(X[i], mean, stdev)

        return probabilities

    # 类别
    def predict(self, x_test):
        # 将预测数据在所有类别中的概率进行排序，并取概率最高的类别
        label = sorted(self.calculate_probabilities(x_test).items(), key=lambda x:
        return label
```

In [272…

```
model = NaiveBayes()
model.fit(train_features.to_numpy(),train_labels.to_numpy())

y_pred = [model.predict(i) for i in test_features.to_numpy()]
# y_pred
```

In [273…

```
precision, recall, F1_score, _ = precision_recall_fscore_support(test_labels, y_pred
acc = accuracy_score(test_labels, y_pred)
print("precision: {} \nrecall: {} \nF1 score: {} \naccuracy: {}".format(precision, r
```

```
precision: [0.89432485 0.41065172]
recall: [0.61980108 0.78342246]
F1 score: [0.73217623 0.53885057]
accuracy: 0.6611486486486486
```