天津大学

机器学习实验报告

题目：机器学习实验---支持向量机

学　　院　　智能与计算学部

专　　业　　人工智能

年　　级　　**2021**

姓　　名　　**666**

学　　号　　我 **TM** 没学会其实

2023 年 4 月 6 日

机器学习实验---贝叶斯

一、实验目的

1. 理解掌握贝叶斯基本思想；
2. 理解掌握极大似然估计基本思想；
3. Python 实现朴素贝叶斯分类；

二、实验内容

1. 基于 sklearn 和 iris 数据集实现朴素贝叶斯分类；
2. 手写代码实现课本的西瓜书数据集。

   数据集：

色泽	根蒂	敲声	纹理	脐部	触感	好瓜
青绿	蜷缩	浊响	清晰	凹陷	硬滑	是
乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	是
乌黑	蜷缩	浊响	清晰	凹陷	硬滑	是
青绿	蜷缩	沉闷	清晰	凹陷	硬滑	是
浅白	蜷缩	浊响	清晰	凹陷	硬滑	是
青绿	稍蜷	浊响	清晰	稍凹	软粘	是
乌黑	稍蜷	浊响	稍糊	稍凹	软粘	是
乌黑	稍蜷	浊响	清晰	稍凹	硬滑	否
乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	否
青绿	硬挺	清脆	清晰	平坦	软粘	否
浅白	硬挺	清脆	模糊	平坦	硬滑	否
浅白	蜷缩	浊响	模糊	平坦	软粘	否
青绿	稍蜷	浊响	稍糊	凹陷	硬滑	否
浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	否
乌黑	稍蜷	浊响	清晰	稍凹	软粘	否
浅白	蜷缩	浊响	模糊	平坦	硬滑	否
青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	否

三、实验报告要求

1. 按实验内容撰写实验过程；
2. 报告中涉及到的代码，每一个模块需要有详细的注释。

四、参考样例

参考样例 1：

```
# encoding=utf-8

import pandas as pd
from sklearn import metrics
```

```
# 加载莺尾花数据集
from sklearn import datasets
# 导入高斯朴素贝叶斯分类器
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

data = datasets.load_iris()
iris_target = data.target #得到数据对应的标签
iris_features = pd.DataFrame(data=data.data,
columns=data.feature_names) #利用 Pandas 转化为 DataFrame 格式
X_train, X_test, y_train, y_test = train_test_split(iris_features,
iris_target, test_size=0.2, random_state=0)
# 使用高斯朴素贝叶斯进行计算
clf = GaussianNB()
clf.fit(X_train, y_train)
# 评估
test_predict = clf.predict(X_test)
print('The accuracy of the NB for Test Set is: %d%%' %
(metrics.accuracy_score(y_test,test_predict)*100))
print(test_predict)
print(y_test)

# 预测
y_proba = clf.predict_proba(X_test[:1])
print(X_test[:1])
print(clf.predict(X_test[:1]))
print("预计的概率值:", y_proba)
```

参考样例 2：

```
# encoding=utf-8

import pandas as pd
import numpy as np


class NaiveBayes:
    def __init__(self):
        self.model = {}  # key 为类别名 val 为字典 PClass 表示该类的该类，
PFeature:{}对应对于各个特征的概率
```

```
    def calEntropy(self, y):  # 计算熵
        valRate = y.value_counts().apply(lambda x: x / y.size)  # 频次汇总，得到各个特征对应的概率
        valEntropy = np.inner(valRate, np.log2(valRate)) * -1
        return valEntropy

    def fit(self, xTrain, yTrain=pd.Series()):
        if not yTrain.empty:  # 如果不传，自动选择最后一列作为分类标签
            xTrain = pd.concat([xTrain, yTrain], axis=1)
        self.model = self.buildNaiveBayes(xTrain)
        return self.model

    def buildNaiveBayes(self, xTrain):
        yTrain = xTrain.iloc[:, -1]

        yTrainCounts = yTrain.value_counts()  # 频次汇总，得到各个特征对应的概率

        yTrainCounts = yTrainCounts.apply(lambda x: (x + 1) / (yTrain.size + yTrainCounts.size))  # 使用了拉普拉斯平滑
        retModel = {}
        for nameClass, val in yTrainCounts.items():
            retModel[nameClass] = {'PClass': val, 'PFeature': {}}

        propNamesAll = xTrain.columns[:-1]
        allPropByFeature = {}
        for nameFeature in propNamesAll:
            allPropByFeature[nameFeature] = list(xTrain[nameFeature].value_counts().index)
        # print(allPropByFeature)
        for nameClass, group in xTrain.groupby(xTrain.columns[-1]):
            for nameFeature in propNamesAll:
                eachClassPFeature = {}
                propDatas = group[nameFeature]
                propClassSummary = propDatas.value_counts()  # 频次汇总 得到各个特征对应的概率
                for propName in allPropByFeature[nameFeature]:
                    if not propClassSummary.get(propName):
                        propClassSummary[propName] = 0  # 如果有属性灭有，那么自动补0
                Ni = len(allPropByFeature[nameFeature])
                propClassSummary = propClassSummary.apply(lambda x: (x + 1) / (propDatas.size + Ni))  # 使用了拉普拉斯平滑
                for nameFeatureProp, valP in propClassSummary.items():
```

```
                    eachClassPFeature[nameFeatureProp] = valP
                retModel[nameClass]['PFeature'][nameFeature] = eachClassPFeature

        return retModel

    def predictBySeries(self, data):
        curMaxRate = None
        curClassSelect = None
        for nameClass, infoModel in self.model.items():
            rate = 0
            rate += np.log(infoModel['PClass'])
            PFeature = infoModel['PFeature']

            for nameFeature, val in data.items():
                propsRate = PFeature.get(nameFeature)
                if not propsRate:
                    continue
                rate += np.log(propsRate.get(val, 0))  # 使用 log 加法避免很小的小数连续乘，接近零
                # print(nameFeature, val, propsRate.get(val, 0))
            # print(nameClass, rate)
            if curMaxRate == None or rate > curMaxRate:
                curMaxRate = rate
                curClassSelect = nameClass

        return curClassSelect

    def predict(self, data):
        if isinstance(data, pd.Series):
            return self.predictBySeries(data)
        return data.apply(lambda d: self.predictBySeries(d), axis=1)

print('start')
dataTrain = pd.read_csv("watermelon_reference.txt", encoding='utf-8', sep=' ')
# dataTrain = pd.read_json("watermelon_reference.txt", encoding="gbk")
print('start')
naiveBayes = NaiveBayes()
treeData = naiveBayes.fit(dataTrain)

import json

print(json.dumps(treeData, ensure_ascii=False))
```

```
pd = pd.DataFrame({'预测值': naiveBayes.predict(dataTrain), '正取值': dataTrain.iloc[:, -1]})
print(pd)
print('正确率:%f%%' % (pd[pd['预测值'] == pd['正取值']].shape[0] * 100.0 / pd.shape[0]))
print('done')
```

五、运行结果

1. 在参考样例 1 中预测测试集上的结果，并输出精度；

```
from sklearn import datasets
# 导入高斯朴素贝叶斯分类器
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

data = datasets.load_iris()
iris_target = data.target #得到数据对应的标签
iris_features = pd.DataFrame(data=data.data, columns=data.feature_names) #利用Pandas转化为DataFrame格式
X_train, X_test, y_train, y_test = train_test_split(iris_features, iris_target, test_size=0.2, random_state=
# 使用高斯朴素贝叶斯进行计算
clf = GaussianNB()
clf.fit(X_train, y_train)
# 评估
test_predict = clf.predict(X_test)
print('The accuracy of the NB for Test Set is: %d%%' % (metrics.accuracy_score(y_test,test_predict)*100))
# print(test_predict)
# print(y_test)

# 预测
y_proba = clf.predict_proba(X_test[:1])
# print(X_test[:1])
# print(clf.predict(X_test[:1]))
# print("预计的概率值:", y_proba)
```

```
The accuracy of the NB for Test Set is: 96%
```

2. 在参考样例 2 中预测测试集上的结果，并输出精度。

```
print('done')
```

```
start
start
正确率:76.470588%
done
```

六、实验小结

本次实验是理解朴素贝叶斯算法的原理并实现。理解极大似然、贝叶斯定理在其中的意义。