数据挖掘第二次作业——实现一个分类器

作业要求

> 请自己编程实现一个分类器（如决策树、朴素贝叶斯、基于规则的分类器等），自行确定实验数据集，在数据集上与商用系统中的同类分类器在不同指标上开展性能对比，并用t检验确定你的分类器和对比的分类器的性能差异是否显著?
>
> ## 实验内容
>
> 本次实验选择了朴素贝叶斯作为学习和实现的对象，有关朴素贝叶斯的理论知识可以参考李航的《统计学习方法》和浙江大学胡浩基老师的《机器学习》课程。
>
> ## 数据集
>
> 本次实验选择了经典的breast_cancer数据集，使用sklearn提供的接口；同样的，也可以自行在UCI的网站上下载。

朴素贝叶斯简介

简介

朴素贝叶斯法是机遇贝叶斯定理与特征条件独立假设的分类方法。 对于给定的训练集，首先机遇特征条件独立假设输入输出的联合概率分布；然后根据此模型，对于给定的输入x，利用贝叶斯定理求出后验概率的最大输出的y。 在了解朴素贝叶斯之前，我们首先需要了解贝叶斯定理。

贝叶斯定理

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_j P(B|A_i)P(A_i)}$$

4.1 朴素贝叶斯的学习与分类

4.1.1 基本方法

训练数据集T。 朴素贝叶斯通过训练数据计算先验概率分布和条件概率分布来学习联合概率分布函数。 这边直接给出最终的表示形式

$$y = \arg\max_{ck} P(Y = c_k) \prod_j P(X^{(j)} = x^{(j)}|Y = c_k)$$

4.2 朴素贝叶斯的参数估计

4.2.1 极大似然估计

这一部分略去。

学习与分类算法

输入：训练数据T 输出：实例x的分类

1. 计算现眼吗概率和条件概率
2. 对于给定的实例x，计算：

$$P(Y=c_k)\prod_{j=1}^{n}P(X^{(j)}=x^{(j)}|Y=c_k)$$

3. 确定x的类为：使得上述值最大的y。

高斯朴素贝叶斯

由于本次实验的数据为连续值，所以我们使用高斯朴素贝叶斯来实现分类操作。 一般使用的模型还有：

1. 多项式模型
2. 伯努利模型

feature的可能性被假设为高斯函数：

$$P\left(x_i \mid y_k\right)=\frac{1}{\sqrt{2\pi\sigma_{yk}^2}}\exp\left(-\frac{\left(x_i-\mu_{yk}\right)^2}{2\sigma_{yk}^2}\right) \quad (1)$$

代码实现

In [121…

```
# 数据处理
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# iris dataset的导入
from sklearn.datasets import load_iris
from sklearn.datasets import load_breast_cancer
# 训练/测试集
from sklearn.model_selection import train_test_split
from collections import Counter
```

In [122…

```
# 数据集初始化
def load_dataset():
    # return X, y(np.array)
    cancer = load_breast_cancer()
    data = pd.DataFrame(cancer.data, columns=cancer.feature_names)
    data['label'] = cancer.target
    data = np.asarray(data)
    return data[:, :-1], data[:, -1]
```

In [123…

```
X, y = load_dataset()
# 训练集和测试集分割
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [124…

```
# 朴素贝叶斯分类器的实现
class Bayes:
    def __init__(self):
        self.core = None

    # 期望
    def _E(self, X):
        return sum(X) / float(len(X))
    # 标准差
    def _std(self, X):
        return np.std(X)

    # 高斯函数计算
    def _gauss(self, x, E, std):
        exp = math.exp(-(math.pow(x - E, 2) / (2 * math.pow(std, 2))))
        return (1 / (math.sqrt(2 * math.pi) * std)) * exp

    # 训练数据的处理
    def _preprocess(self, data):
        # return [(E, std)***]
        # feed for the _gauss() function
        return [(self._E(i), self._std(i)) for i in zip(*data)]

    # 训练模型：求出每个标签的数学期望和标准差
    def fit(self, X, y):
        labels = list(set(y))
        data = {l: [] for l in labels}
        for arr, l in zip(X, y):
            data[l].append(arr)
        self.core = {
            l: self._preprocess(v)
            for l, v in data.items()
        }

    # 计算概率
    def _calculate(self, data):
        prob = {}
        for label, values in self.core.items():
            temp = 1
            for i, value in enumerate(values):
                E = value[0]
                std = value[1]
                temp *= self._gauss(data[i], E, std)
            prob[label] = temp
        return prob



    # 预测函数
    def predict(self, X):
        # 返回概率最大的label
        return sorted(self._calculate(X).items(), key=lambda x: x[1], reverse

    # 评价函数：计算accu
    def score(self, X, y):
        count = 0
        for x, label in zip(X, y):
            y_pred = self.predict(x)
            if y_pred == label:
                count += 1
        return float(count)/float(len(y))
```

In [141…
```
model = Bayes()
```

In [142…
```
model.fit(X_train, y_train)
```

In [143…
```
# accu
model.score(X_test, y_test)
```

Out[143… 0.9210526315789473

sklearn 高斯贝叶斯

接下来部分，我们展现sklearn中实现的高斯贝叶斯分类器

In [144…
```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
```

In [145…
```
clf.fit(X_train, y_train)
```

Out[145… GaussianNB()

In [146…
```
# accu
clf.score(X_test, y_test)
```

Out[146… 0.9298245614035088

性能检验

在此部分我们使用了accuracy, binary-F1在进行测试，均进行25次测试，使用 `scipy.stats` 来进行t检验。 我们在25次测试中，通过设置 `train_test_split` 中 `random_state` 的值为 `None` 来确保每次生成的训练/测试数据不同。

此处的F1指的是：F1 = 2 *(precision* recall) / (precision + recall), 是相对于precision和recall而言更加综合的指标。

In [149…
```
# accuracy测试

myAccu = []
for _ in range(1):
    myModel = Bayes()
    myModel.fit(X_train, y_train)
    print(myModel.score(X_test, y_test))

skAccu = []
for _ in range(25):
    skModel = GaussianNB()
    skModel.fit(X_train, y_train)
    skAccu.append(skModel.score(X_test, y_test))
```

0.9210526315789473

In [169…

```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
import sklearn.metrics as metrics
```

In [161…

```
myAccu = []
myF1 = []
skAccu = []
skF1 = []
for _ in range(25):
    # my model
    tx_train, tx_test, ty_train, ty_test = train_test_split(X, y, random_stat
    myModel = Bayes()
    myModel.fit(tx_train, ty_train)
    myPred = [myModel.predict(x) for x in tx_test]
    myAccu.append(myModel.score(tx_test, ty_test))
    myF1.append(f1_score(ty_test, myPred))

    # sklearn
    skModel = GaussianNB()
    skModel.fit(tx_train, ty_train)
    skPred = skModel.predict(tx_test)
    skAccu.append(skModel.score(tx_test, ty_test))
    skF1.append(f1_score(ty_test, skPred))
```

In [167…

```
print("My F1 score: "+str(np.mean(myF1)))
print("My Accu is: "+str(np.mean(myAccu)))

print("SK F1 score: "+str(np.mean(skF1)))
print("SK Accu is: "+str(np.mean(skAccu)))
```

```
My F1 score: 0.948714020512844
My Accu is: 0.9348251748251749
SK F1 score: 0.953743031963181
SK Accu is: 0.9406993006993009
```

从结果可以看出，无论是精确度还是F1，sklearn提供的高斯朴素贝叶斯的效果都是比我自己实现的要更加好。

t-检验

In [165…

```
# 进行t-检验
from scipy import stats
stats.ttest_rel(myF1, skF1)
```

Out[165… `Ttest_relResult(statistic=-4.772492343041904, pvalue=7.405689255925076e-05)`

原假设： H0:μ=μ0 H1:μ≠μ0 所以拒绝原假设。

所以总体而言的话sklearn自带的高斯朴素贝叶斯的整体性能高于我自己实现的模型。