实验三 分类

1. 准确率

|  |  |  |
| --- | --- | --- |
| C4.5 | 朴素贝叶斯 | KNN |
| 0.64 | 0.69 | 0.65 |

1. 优缺点

|  |  |  |
| --- | --- | --- |
|  | 优点 | 缺点 |
| C4.5 | 训练需要的数据少；便于理解和解释。 | 有时是不稳定的，因为数据中的微小变化可能会导致完全不同的树生成。且容易出现过拟合。 |
| 朴素贝叶斯 | 具有很好的数学原理。而且在数据量很小的时候表现良好，数据量很大的时候也可以进行增量计算，效率稳定。 | 在属性个数比较多或者属性之间相关性较大时，分类效果不好。 |
| KNN | 简单，易于理解。特别适合于多分类问题，还可以处理回归问题，也就是预测。 | 计算量太大，尤其是特征数非常多的时候。每一个待分类文本都要计算它到全体已知样本的距离，才能得到它的第K个最近邻点。 |

1. 预处理

创建一个类，输入为xlsx文件路径read\_path和训练集占比train\_rate。对数据进行01二元化，并且随机抽取train\_rate\*100%的数据作为训练集，训练集的销量高低比约等于原数据的销量高低比。并将剩余的数据作为测试集。

class C45():

    train\_data = None

    train\_x = None

    train\_y = None

    test\_data = None

    test\_x = None

    test\_y = None

    train\_rate = 0.8

    rp\_in = ['好','坏', '是', '否', '高', '低']

    rp\_v = [1, 0, 1, 0, 1, 0]

    clf = tree.DecisionTreeClassifier()

    def \_\_init\_\_(self, read\_path, train\_rate):

        Input = pd.read\_excel(read\_path)

        self.train\_rate = train\_rate

        Input = self.input\_pre(Input)

        self.data\_init(Input)

    def input\_pre(self, Input):

        return Input.replace(self.rp\_in, self.rp\_v)

    def data\_init(self,Input:pd.DataFrame):

        count = Input.iloc[:,4:].value\_counts()

        high = int(count[1]\*self.train\_rate)

        low = int(count[0]\*self.train\_rate)

        high\_data = Input[Input['销量'] == 1].reset\_index(drop=True)

        low\_data = Input[Input['销量'] == 0].reset\_index(drop=True)

        self.train\_data = pd.concat([high\_data.sample(n=high), low\_data.sample(n=low)], ignore\_index=True).reset\_index(drop=True)

        self.test\_data = pd.concat([Input, self.train\_data]).drop\_duplicates(keep=False).reset\_index(drop=True)

        key = Input.keys()

        order = [k for k in key[1:4]]

        # order.insert(0, key[4])

        self.train\_x = np.array(self.train\_data[order])

        self.train\_y = np.array(self.train\_data[key[4]])

        self.test\_x = np.array(self.test\_data[order])

        self.test\_y = np.array(self.test\_data[key[4]])

1. C4.5决策树

调用sklearn库的决策树，重复训练测试100次，后两种算法也是如此。最后精确度平均值为0.64

if \_\_name\_\_ == '\_\_main\_\_':

    ac = 0.0

    times = 100

    for i in range(0,times):

        C = C45("./sales\_data.xls",0.8)

        # print(f'train\_x={C.train\_x}')

        # print(f'train\_y={C.train\_y}')

        # print(f'train\_x={C.test\_x}')

        # print(f'train\_y={C.test\_y}')

        C.clf = C.clf.fit(C.train\_x, C.train\_y)

        if i==0:

            dot\_data = tree.export\_graphviz(C.clf, out\_file=None)

            graph = graphviz.Source(dot\_data)

            graph.render("iris")

        pre = np.array(C.clf.predict(C.test\_x))

        real = np.array(C.test\_y)

        # print(f'pre = {pre}')

        # print(f'real = {real}')

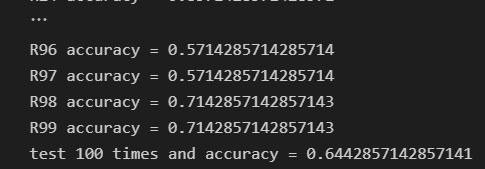
        ac\_tmp = np.sum(pre==real)/pre.size

        ac += ac\_tmp

        print(f'R{i} accuracy = {ac\_tmp}')

    ac /= times

    print(f"test {times} times and accuracy = {ac}")



1. 朴素贝叶斯

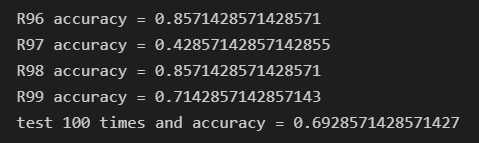
精确度平均值为0.69

By = Bayes("./sales\_data.xls",0.8)

    By.gnb = By.gnb.fit(By.train\_x, By.train\_y)

    pre = np.array(By.gnb.predict(By.test\_x))

    real = np.array(By.test\_y)



1. KNN分类器

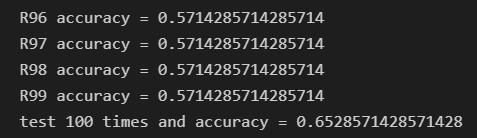
精确度平均值为0.65

        K = KNN("./sales\_data.xls",0.8)

        K.neigh = K.neigh.fit(K.train\_x, K.train\_y)

        pre = np.array(K.neigh.predict(K.test\_x))

        real = np.array(K.test\_y)



1. 除了分类准确率，速度和系统资源消耗也可以结合用来衡量一个分类器的性能。系统资源如内存占用、cpu占用等等。