重庆邮电大学

学生实验实习报告册

学年学期：	2019 -2020 学年第二学期
课程名称：	数据挖掘基础A线上实验
学生学院：	计算机科学与技术学院
专业班级：	智能科学与技术/2班与3班
学生学号：	2017212019， 2017212072，2017211751
学生姓名：	李彬楷，陈臻，汤世展
联系电话：	13206032678

重庆邮电大学教务处印制

教师评阅记录表

【重要说明】

- 学生提交报告册最终版时，**必须包含此页**，否则不予成绩评定。
- 本报告册模板内容格式除确实因为填写内容改变了布局外，**不得变更其余部分的格式**，否则不予成绩评定。

报告是否符合考核规范	☑ 符合　☐ 不符合
报告格式是否符合标准	☑ 符合　☐ 不符合
报告是否完成要求内容	☑ 是　☐ 否
报告评语：	
报告成绩：	
评阅人签名（签章） 年　月　日	

目录

实验题目

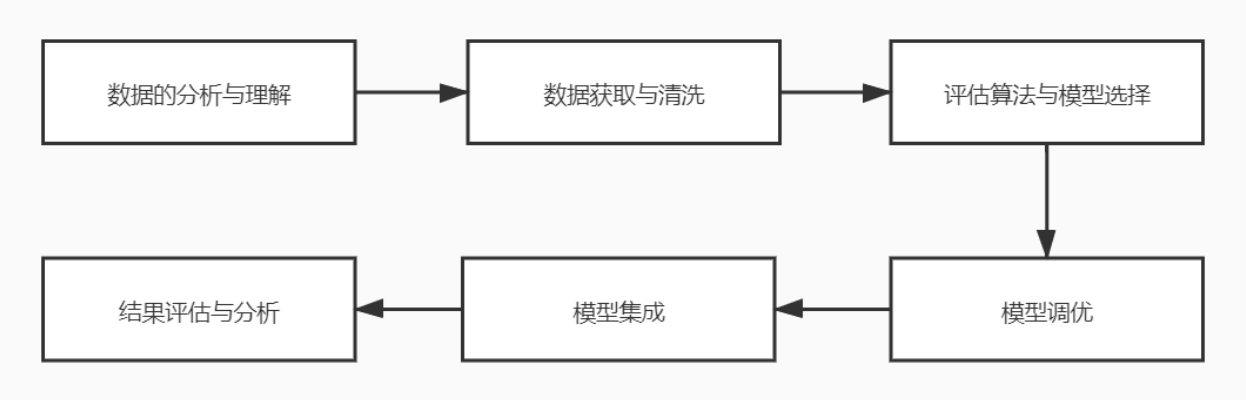
- 本实验采用 UCI 中的 mushroom 数据集
- 数据集链接：https://archive.ics.uci.edu/ml/datasets/Mushroom

需求分析

- 本数据集摘自《奥杜邦学会野外指南》，需求是根据蘑菇的物理特性描述对蘑菇进行分类：有毒或食用。每一种蘑菇都被确定为绝对可食用，绝对有毒，或未知的可食用性，不推荐食用。后一类与有毒的一类结合在一起，故该问题为二分类问题。

任务总体设计

- 整体流程图



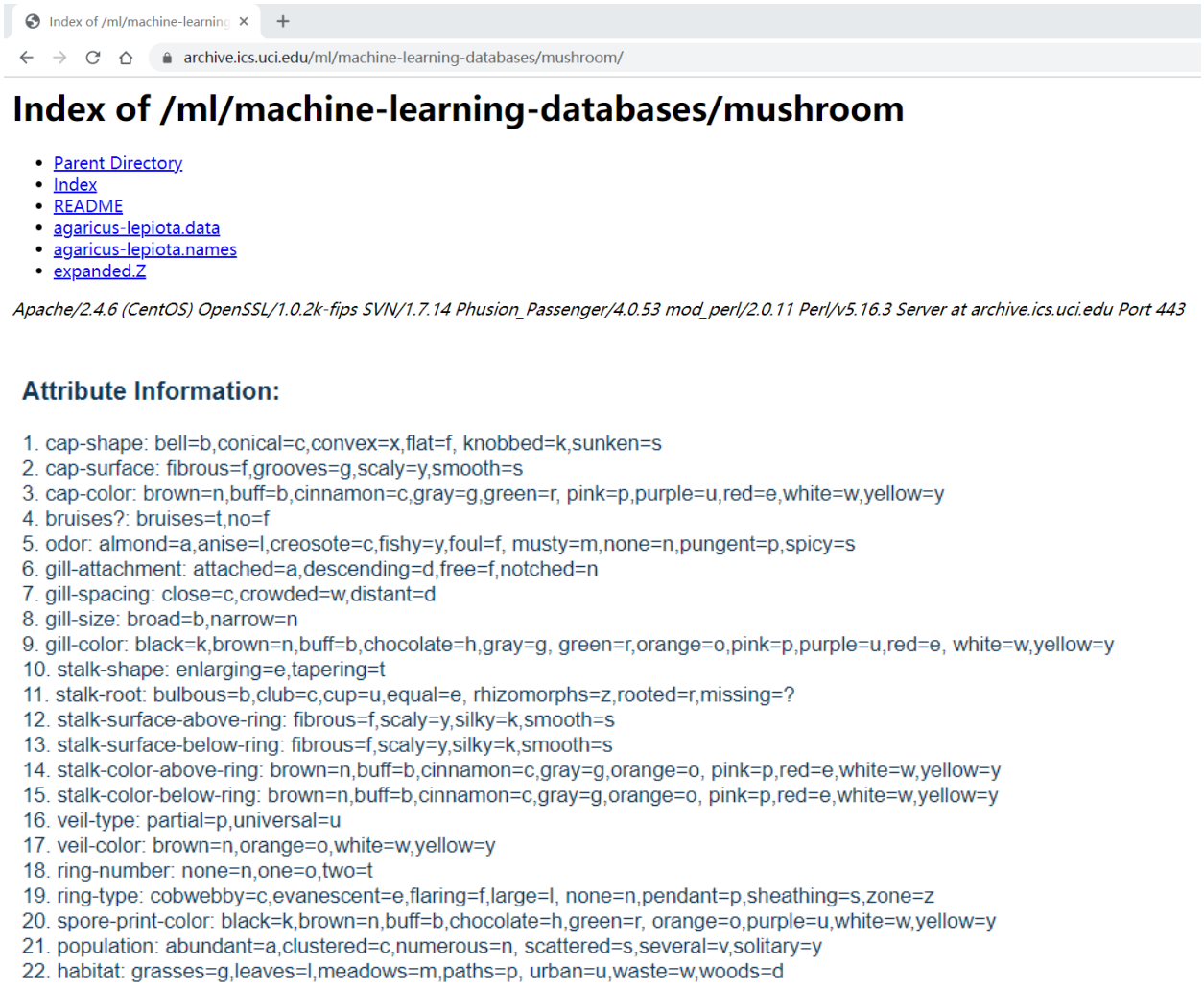
- 具体步骤

1. 数据分析与理解：通过直方图、相关性矩阵等可视化的方式发现数据的初步规律
2. 数据获取与清洗：通过 ASCII 编码将字符将数据集转换为数字的形式便于后续的模型训练，并将通过第一步发现的无用数据以及缺失数据删除
3. 评估算法与模型选择：采取 10 折交叉验证对 KNN、决策树、朴素贝叶斯、SVM、LDA 算法进行评估，并通过箱线图进行结果的可视化展示
4. 模型调优：通过正态化等方法处理数据，有效地提高了 SVM 的分类能力
5. 模型集成：通过面向对象的思想将各个模型对外调用方法统一起来，选取 BP 全连接神经网络、KNN、决策树进行模型的集成
6. 对集成算法的结果进行评估与思考

任务详细设计

问题定义

- 本实验数据来自于 UCI 的 mushroom 数据集，agaricus-lepiota.data 为数据与标签，agaricus-lepiota.names 为元数据。我们需要通过 agaricus-lepiota.data 中的部分数据进行模型的训练，然后使用部分数据进行模型的验证与评估。



导入数据

- agaricus-lepiota.data 中的数据如图所示（其中？为缺失的数据），可见其为标称属性（形式为字符），为了方便数据可视化，我们需要将其转换为对应的数字类型。

```
8121    e,x,s,n,f,n,a,c,b,y,e,?,s,s,o,o,p,n,o,p,b,v,l
8122    e,f,s,n,f,n,a,c,b,n,e,?,s,s,o,o,p,o,o,p,b,c,l
8123    p,k,y,n,f,y,f,c,n,b,t,?,s,k,w,w,p,w,o,e,w,v,l
8124    e,x,s,n,f,n,a,c,b,y,e,?,s,s,o,o,p,o,o,p,o,c,l
8125
```

- 下面先是通过读取 csv 文件然后，通过字符对应的 ASCII 值将其转换为数字类型，再将处理过的数据保存为另外的数据文件 data_preceded.csv。此外，通过分析数据发现，“stalk-root”对应的数据列是唯一存在缺失的数据列，所以在读取数据之后将其去除。

```
def __char_to_int():
    names = get_names()
    df = read_csv('../../data/agaricus-lepiota.data', names=names)
    # 去掉缺失值多的一列
    df.drop('stalk-root', axis=1, inplace=True)
    # print(df.shape)
    dataSet = []
    for d in df._values:
        data = []
        for cidx in range(len(d)):
            # 标签
            if cidx == 0:
                if d[cidx] == 'p':
                    data.append(0)
                else:
                    data.append(1)
            # 数据
            else:
                data.append(ord(d[cidx]) - ord('a'))
        dataSet.append(data)
    result = DataFrame(dataSet, columns=df.keys())
    f = open('../../data/data_preceded.csv', 'w')
    writer = csv.writer(f)
```

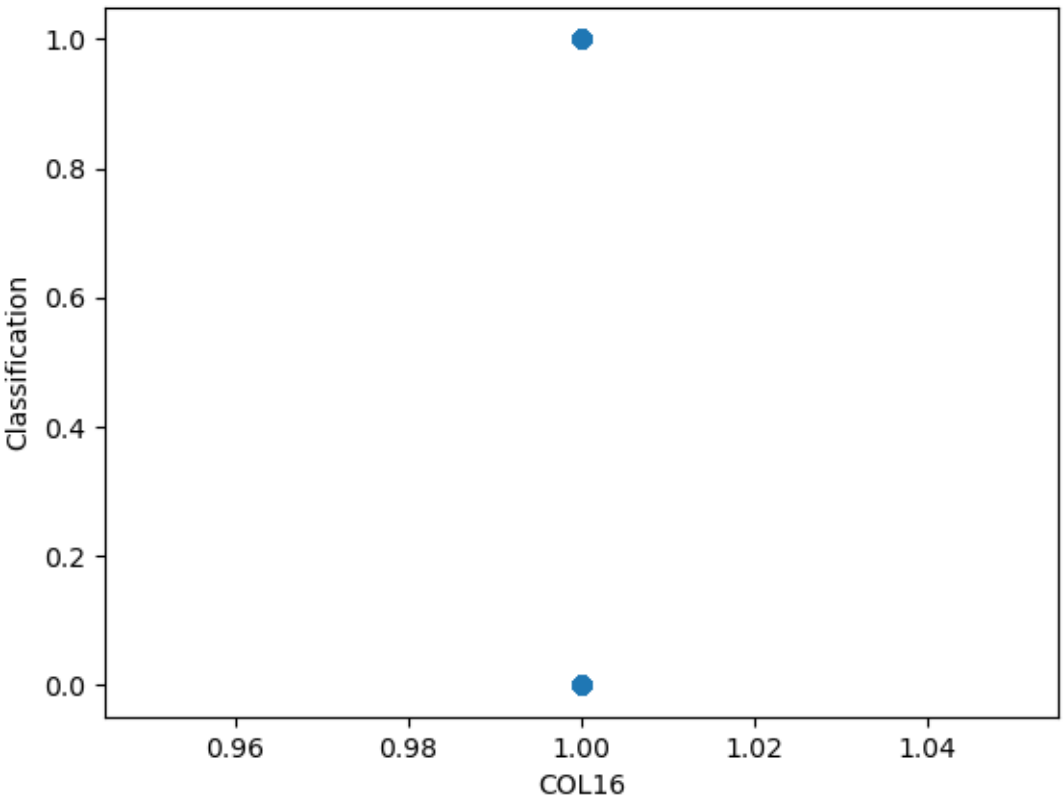
```
    writer.writerow(result.keys())
    writer.writerows(result.values)
```

- 通过读取上述处理过的数据文件，可以获取到处理后的数据

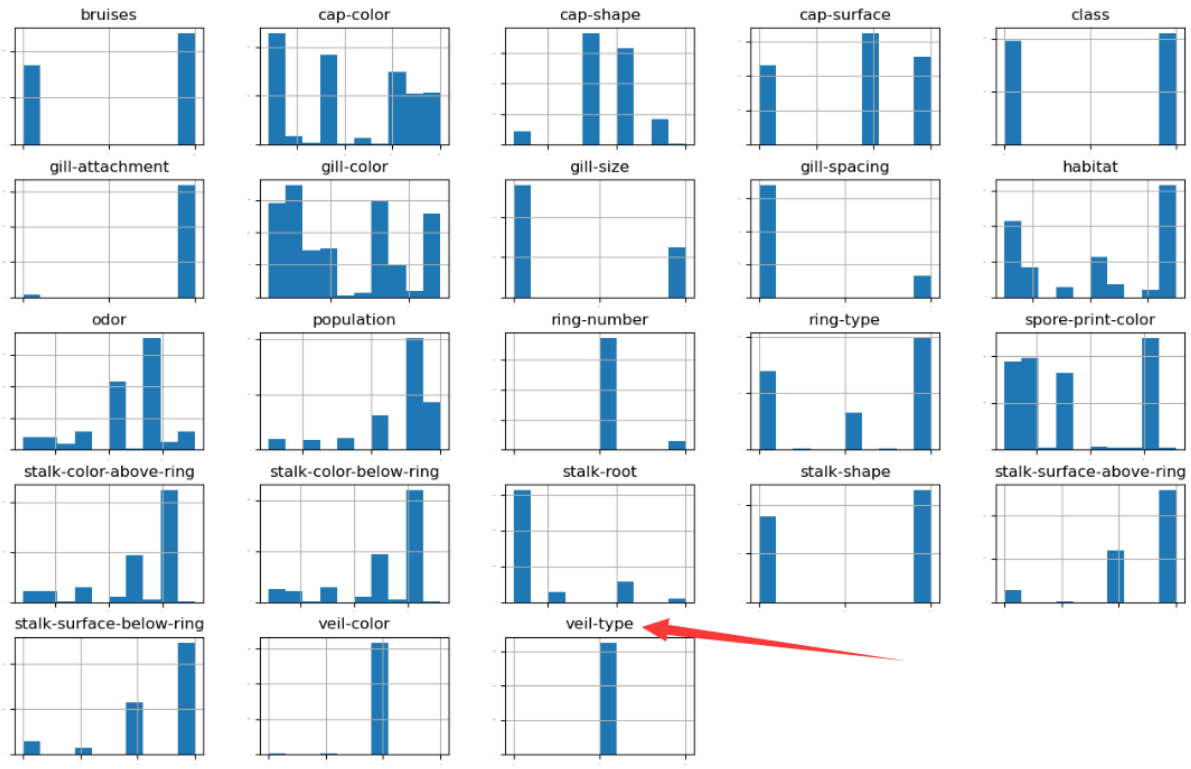
```
def get_total_data():
    data = read_csv('../../data/data_preceded.csv')
    return data
```

分析数据

- 通过散点图可以看到数据每一个维度与类别之间的关系，如下图第 16 个属性 veil-type，无论类别是那种，其取值均不变，所以该属性可以去除。

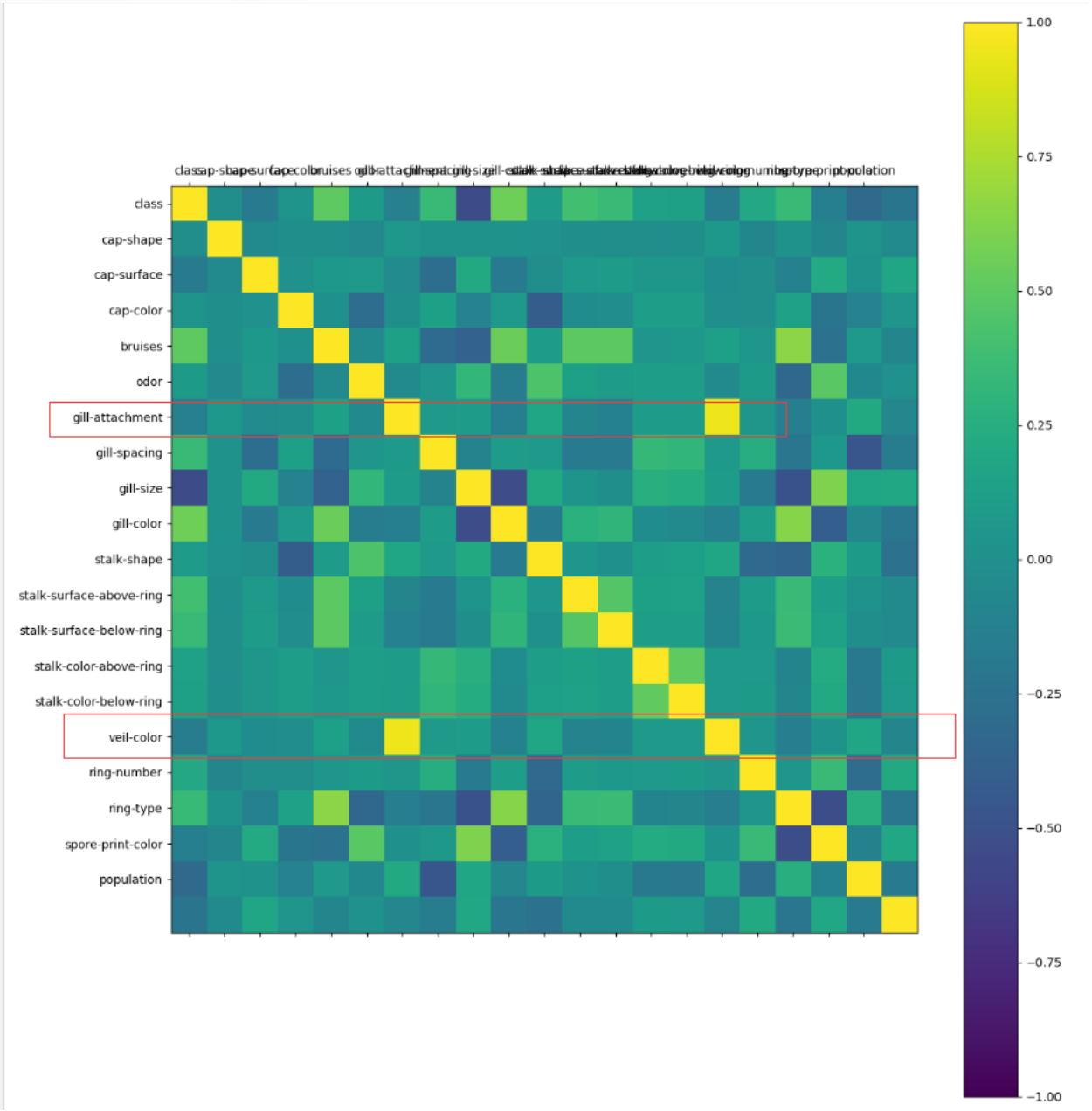


- 通过直方图查看数据的分布情况，可以看到 veil-type 对应的数据列中的数据全部是同样的值，对数据分析没有意义，所以后续可以直接将其去除。此外，可以看到数据分布不是特别均匀，后续可以进行正态化处理。




```
def histogram_visual():
    data = get_total_data()
    data.hist(sharex=False, sharey=False, xlabelsize=1, ylabelsize=1,
figsize=(16, 10))
    plt.show()
```

- 通过数据相关矩阵图可以看到数据之间的相关性，可以看到 veil-color 与 gill-attachment 两个特征之间具有正相关关系，gill-color 与 gill-size 有负相关关系。




```
def figure(data):
    fig = plt.figure(figsize=(13, 13))
    ax = fig.add_subplot(111)
    cax = ax.matshow(data.corr(), vmin=-1, vmax=1, interpolation='none')
    fig.colorbar(cax)
    # 刻度
    ticks = np.arange(0, 20, 1)
    ax.set_xticks(ticks)
    ax.set_yticks(ticks)
    names = list(data.columns)
```

```
    ax.set_xticklabels(names)
    ax.set_yticklabels(names)
    plt.show()
```

- 使用箱线图对模型的 k 折交叉验证进行可视化展示（后续使用模块）

```
import matplotlib.pyplot as plt
def box_plot(results, names):
    fig = plt.figure()
    fig.suptitle('AlgorithmComparison')
    ax = fig.add_subplot(111)
    plt.boxplot(results)
    ax.set_xticklabels(names)
    plt.show()
```

分离评估数据集

- 通过切分数据集将数据分为训练集以及测试集

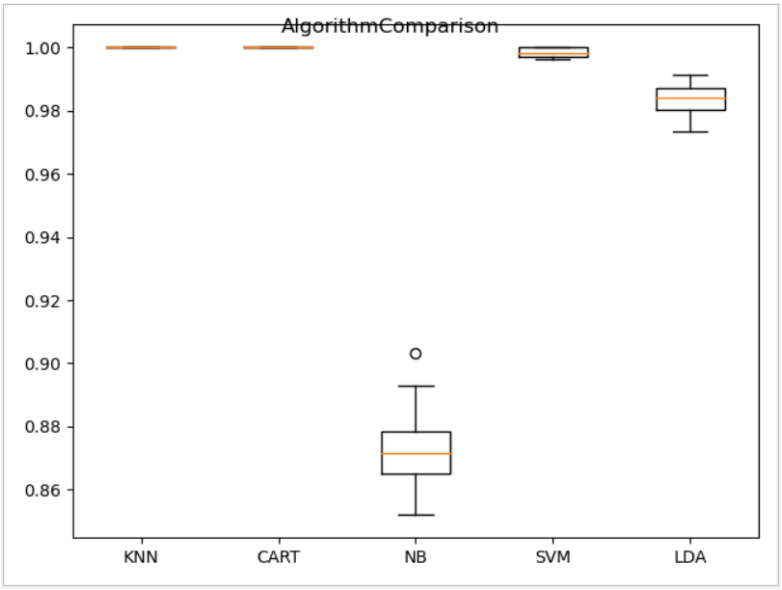
```
from sklearn.model_selection import train_test_split
def data_split():
    data_set = get_total_data()
    arr = data_set.values
    x = arr[:, 1:arr.shape[1]]
    y = arr[:, 0]
    test_size = 0.3
    seed = 7
    # train_x,test_x,train_y,test_y
    return train_test_split(x, y, test_size=test_size, random_state=seed)
```

评估算法

- 采用 sklearn 中的 KNN、决策树、朴素贝叶斯、SVM 以及 LDA 使用没有经过任何处理的数据训练集进行训练作为基准模型，采取 10 折交叉验证对模型进行评估。可以看到，除了朴素贝叶斯模型之外，其余模型准确率非常高，特别是 KNN 与决策树，准确率达

到了 100%。

```
KNN 1.000000 (0.000000)
CART 1.000000 (0.000000)
NB 0.873019 (0.015314)
SVM 0.998417 (0.001461)
LDA 0.983820 (0.005087)
```



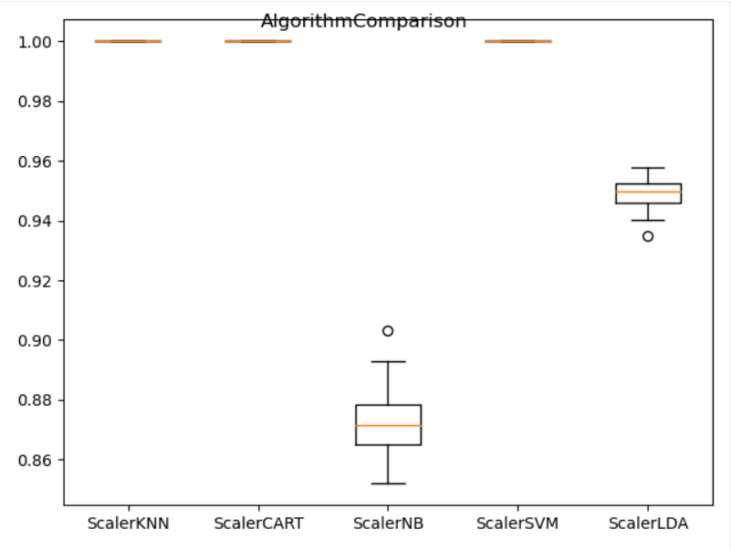

```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

num_folds = 10
seed = 7
scoring = 'accuracy'

def baseline(train_x, train_y):
    models = {}
    models['KNN'] = KNeighborsClassifier()
    models['CART'] = DecisionTreeClassifier()
    models['NB'] = GaussianNB()
    models['SVM'] = SVC()
    models['LDA'] = QuadraticDiscriminantAnalysis()
    results = []
```

```
    for key in models:
        fold = KFold(n_splits=num_folds, random_state=seed, shuffle=True)
        result = cross_val_score(models[key], train_x, train_y, cv=fold,
scoring=scoring)
        results.append(result)
        print("%s %f (%f)" % (key, result.mean(), result.std()))
        # print(result)
    box_plot(results, names=models.keys())
```

算法调参

- 对上述模型进行优化，主要是使用 Pipeline 流水线对数据进行正态化处理。可以看到正态化数据之后，SVM 模型的准确率也提升到了 100%。

```
ScalerKNN 1.000000 (0.000000)
ScalerCART 1.000000 (0.000000)
ScalerNB 0.873019 (0.015314)
ScalerSVM 1.000000 (0.000000)
ScalerLDA 0.948294 (0.006356)
```




```
num_folds = 10
seed = 7
scoring = 'accuracy'

def scaler(train_x, train_y):
    piplelines = {}
    piplelines['ScalerKNN'] = Pipeline([('Scaler', StandardScaler()), ('KNN',
KNeighborsClassifier())])
    piplelines['ScalerCART'] = Pipeline([('Scaler', StandardScaler()),
```

```
('CART', DecisionTreeClassifier())])
    piplelines['ScalerNB'] = Pipeline([('Scaler', StandardScaler()), ('NB',
GaussianNB())])
    piplelines['ScalerSVM'] = Pipeline([('Scaler', StandardScaler()), ('SVM',
SVC())])
    piplelines['ScalerLDA'] = Pipeline([('Scaler', StandardScaler()), ('LDA',
LinearDiscriminantAnalysis())])
    results = []
    for key in piplelines:
        fold = KFold(n_splits=num_folds, random_state=seed, shuffle=True)
        result = cross_val_score(piplelines[key], train_x, train_y, cv=fold,
scoring=scoring)
        results.append(result)
        print("%s %f (%f)" % (key, result.mean(), result.std()))
        # print(result)
    box_plot(results, names=piplelines.keys())
```

利用面向对象思想进行协作开发

- 协作开发中，代码的规范性是非常重要。所以我们定义了一个分类器基类 BaseClassifier，具体的分类器通过继承该基类并重写父方法，这样会便于后续的模型的集成。

```
# 分类器基类，每个分类器均需要继承该基类，便于后续的组合
class BaseClassifier:
    # 训练 输入 train_data_x 以及 train_data_y，无返回值
    def train(self, train_data_x, train_data_y):
        pass

    # 分类方法 输入一个 n*1 的向量，输出 test_data_y 即分类标签
    def classify(self, test_data_x):
        pass
```

- 继承与重写示例

1. 搭建包含一层隐含层的 BP 全连接神经网络分类模型 NeuralNetwork

```
class NeuralNetwork:
    threshold = 26

    def __init__(self, layer_num, learn_step, neuron_num_each_layer):
        # 网络层数，包括输出层
        self.layer_num = layer_num
        # 学习率
        self.learn_step = learn_step
        # 每层的神经元数目（一个 list）
        self.neuron_num_each_layer = neuron_num_each_layer
        # 激活函数
        self.active_function = lambda x: 1.0 / (1.0 + np.exp(-x))
        # 神经网络的全部权值都保存于此
        self.weight = []
        for i in range(layer_num):
            # 生成[0,1)之间的数据
            self.weight.append(np.random.random((self.neuron_num_each_layer[i
+ 1], self.neuron_num_each_layer[i])))

    def update(self, train_x, train_y):
        train_x = train_x / 26
        temp = np.zeros(2)
        temp[train_y] = 1
        train_y = np.array(temp)
        inputs = np.array(train_x, ndmin=2).T
        targets = np.array(train_y, ndmin=2).T
        # 正向传播
        self.outputs = []
        # 输入层的输出就是原始输入
```

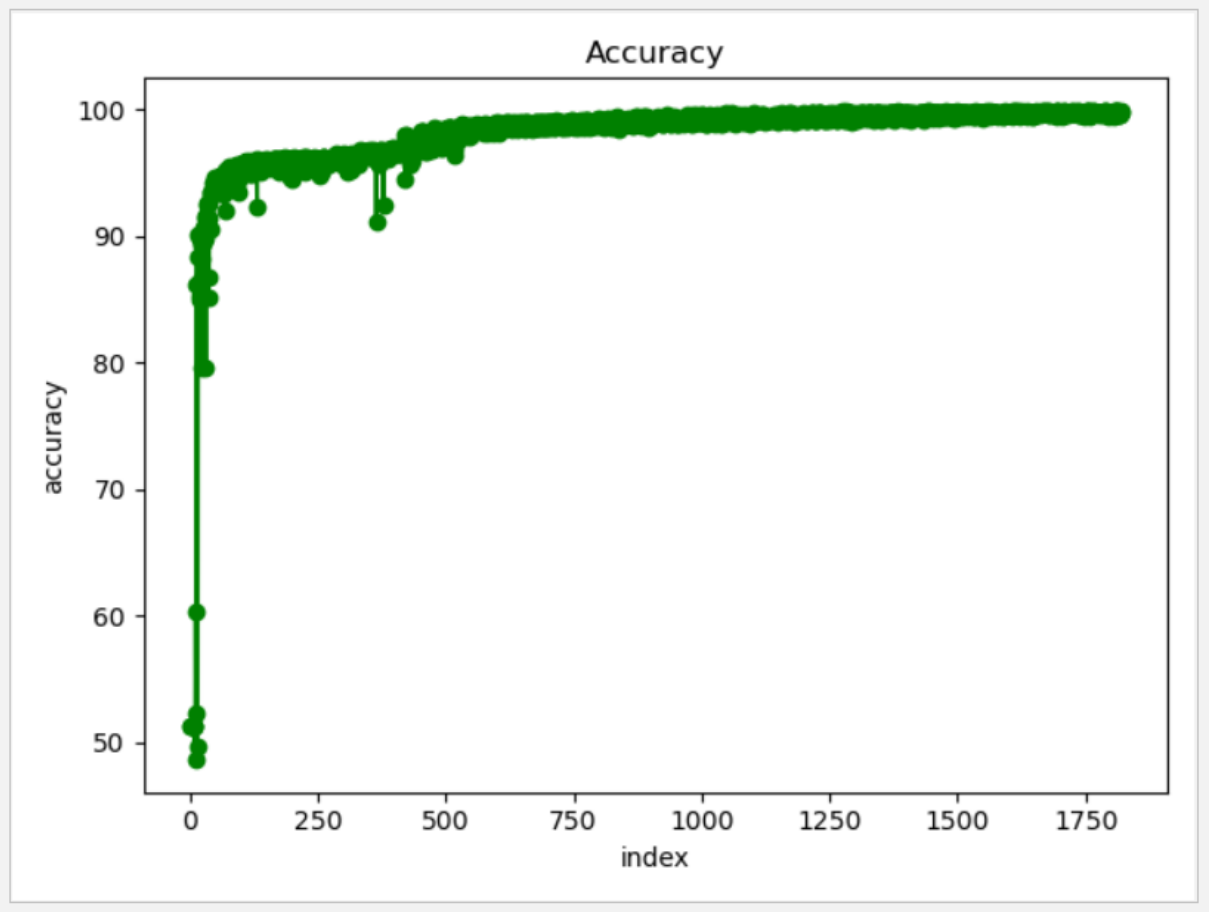
```
        self.outputs.append(inputs)
        for i in range(self.layer_num):
            temp_inputs = np.dot(self.weight[i], inputs)
            temp_outputs = self.active_function(temp_inputs)
            # 当前层的输出是下一层的输入
            inputs = temp_outputs
            self.outputs.append(temp_outputs)
        # 计算误差
        self.output_errors = []
        for i in range(self.layer_num):
            if i == 0:
                # 输出层的误差=目标值-输出值
                self.output_errors.append(targets - self.outputs[-1])
            else:
                # 隐层的误差=当前隐层与下一层之间的权值矩阵与下一层误差矩阵的乘积
                self.output_errors.append(np.dot((self.weight[self.layer_num - i]).T, self.output_errors[i - 1]))
        # print("LOSS:", np.sum(self.output_errors[-1]))
        # 反向传播
        for i in range(self.layer_num):
            # f(x)* （1-f(x)）即为激活函数 f(x)的导函数，更新过程从后向前进行
            self.weight[self.layer_num - i - 1] += self.learn_step * np.dot(
                (self.output_errors[i] * self.outputs[-1 - i] * (1.0 - self.outputs[-1 - i])),
                np.transpose(self.outputs[-1 - i - 1]))

    def test(self, test_x, test_y):
        inputs = np.array(test_x, ndmin=2).T
        for i in range(self.layer_num):
            temp_inputs = np.dot(self.weight[i], inputs)
```

```
            temp_outputs = self.active_function(temp_inputs)
            inputs = temp_outputs
        # 判断输出层最接近 1 的那个神经元的下标是否与标签中为 1（一组标签只有一个 1）的那个下标一致
        return list(inputs).index(max(list(inputs))) == list(test_y).index(1)

    def classify(self, test_x):
        test_x = test_x / 26
        inputs = np.array(test_x, ndmin=2).T
        for i in range(self.layer_num):
            temp_inputs = np.dot(self.weight[i], inputs)
            temp_outputs = self.active_function(temp_inputs)
            inputs = temp_outputs
        return list(inputs).index(max(list(inputs)))
```

2. 定义 BPClassifier 并继承自 BaseClassifier，重写对应的方法

```
class BPClassifier(BaseClassifier):
    def __init__(self):
        self.train_times = 8
        learn_step = 0.1
        layers = [20, 40, 2]
        self.network = NeuralNetwork(2, learn_step, layers)

    def train(self, train_data_x, train_data_y):
        for i in range(self.train_times):
            for j in range(len(train_data_x)):
                self.network.update(train_data_x[j], train_data_y[j])

    def classify(self, test_data_x):
        return self.network.classify(test_data_x)
```

3. 可以看到，模型被成功构建并且分类效果较好。尽管模型内部实现复杂，但是通过重写父类的方法，可以对外提供一个简单的调用入口。



算法集成

- 集成算法可以将多个分类器集成在一起，这样可以整体提高准确率以及抗干扰能力。下面我们基于投票的方式集成前面效果较好的模型——KNN、决策树、以及 BP 神经网络。

1. KNN 分类器

```
class KnnClassifier(BaseClassifier):
    def __init__(self):
        self.classifier = KNeighborsClassifier(n_neighbors=8,
algorithm='auto')

    def train(self, train_data_x, train_data_y):
        self.classifier.fit(train_data_x, train_data_y)

    def classify(self, test_data_x):
        return self.classifier.predict(test_data_x)[0]
```

2. 决策树分类器

```
class MyDecisionTreeClassifier(BaseClassifier):
    def __init__(self):
        self.classifier = DecisionTreeClassifier()

    def train(self, train_data_x, train_data_y):
        self.classifier.fit(train_data_x, train_data_y)

    def classify(self, test_data_x):
        return self.classifier.predict(test_data_x)[0]
```

3. 模型集成

```
if __name__ == '__main__':
    train_x, test_x, train_y, test_y = data_split()
    classifiers = {}
    classifiers['BPNetWork'] = BPClassifier()
    classifiers['CART'] = MyDecisionTreeClassifier()
    classifiers['KNN'] = KnnClassifier()
    # 训练与构建模型
    for key in classifiers:
        classifiers[key].train(train_x, train_y)
    print('模型训练完毕...')
    correct_num = 0
    for i in range(len(test_y)):
        test_data_x = test_x[i]
        test_data_y = test_y[i]
        output = []
        for key in classifiers:
            val = classifiers[key].classify(test_data_x.reshape(1, 20))
            output.append(val)
        print('模型输出', output, '测试数据标签', test_data_y)
        # 求出出现次数最多的数字
```

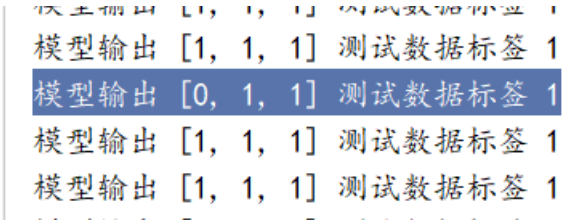
```
        result = max(set(output), key=output.count)
        if result == test_data_y:
            correct_num += 1
    print("正确率：%f%%" % (correct_num * 100 / test_x.shape[0]))
```

- 结果分析

1. 因为我们选择的是通过前面分析出来的，表现优秀的几个模型，所以最终准确率达到了100%。

```
模型输出 [1, 1, 1] 测试数据标签 1
模型输出 [1, 1, 1] 测试数据标签 1
模型输出 [1, 1, 1] 测试数据标签 1
正确率：100.000000%

Process finished with exit code 0
```

2. 同时通过日志输出可以看到，在实际分类过程中，其实有的模型是出现了分类错误的情况的。但是由于集成算法的存在，这种个别模型的分类错误有效地被减低了。



总结

在本实验中，我们通过数据挖掘的流程，对数据进行预处理之后，通过交叉验证方法计算模型的平均分类准确率，将表现优异的模型进行集成，对 UCI 的 mushroom 数据集的分类问题达到了比较好的效果。

心得体会

李彬楷

- 对于输入数据的标准化的预处理这个步骤，对于神经网络来说是非常重要的。如果不进行标准化，会导致输入层输出有可能十分不平衡，这会导致神经网络的权值调整过程要么过大要么过小，导致模型的训练不能完成。

- 对于这个数据集，单个模型如 KNN、决策树的分类准确率已经非常高了（相比之下，朴素贝叶斯算法准确率不高），这出乎我们的意料。但是我们去网站上搜索了一下别人的模型，发现其准确率也非常高。所以我们觉得准确率高的原因是：问题输出较为简单，是二分类问题；数据集的数量多以及分布合理，对于模型的准确率也有好的影响。

陈臻

- 本次课程分析数据的过程，远远比实现几个算法重要，在真正的数据分析项目中，整体的把握往往会对数据分析的结果有意外的提升。数据的预处理也是很重要，是数据分析的基础，良好的数据，才能得到有价值的分析结果。
- 对于这次实验，我自己写了一个 KNN，和 PCA 的算法，可能代码有误，导致 KNN 的训练效果不如意，后来调用官方的库，结果要好得多，官方的库有更好的优化，会对整个算法产生影响。我觉得还需要多学习各类的算法，不能在一棵树上吊死。

汤世展

- 最初我是通过 Excel 来将字符型的数据转换为数字型数据的，后面发现准确率很高，一度怀疑是数据清洗部分出了问题，所以舍弃了人工操作的方式，通过程序编码利用字符的 ASCII 编码进行数据类型的转换。
- 对于这一次数据挖掘实验，我们运用到了课程上学到的知识，按照数据获取，数据清理，特征选择，特征提取，模型选择，模型训练，模型应用这几个步骤对数据进行了挖掘，巩固了知识。其中有用众数补充空缺，手动清理无用属性，对数据进行了 PCA 分析。
- 对于这个数据集，我们选择了多个简单的分类算法，虽然效果已经很好了，为了更加加强分类器的能力，我们经过讨论，采用了合成分类器的思想，使分类器更加准确。本次实验，让我的编程能力得到了锻炼，虽然字写的决策树代码效果不如调包，但是还是得到了锻炼。