

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

[教师评阅记录表 2](#_Toc42264322)

[实验题目 2](#_Toc42264323)

[需求分析 2](#_Toc42264324)

[任务总体设计 2](#_Toc42264325)

[任务详细设计 2](#_Toc42264326)

[问题定义 3](#_Toc42264327)

[导入数据 3](#_Toc42264328)

[分析数据 3](#_Toc42264329)

[描述性统计 3](#_Toc42264330)

[数据可视化 3](#_Toc42264331)

[分离评估数据集 3](#_Toc42264332)

[评估算法 3](#_Toc42264333)

[算法调参 3](#_Toc42264334)

[算法集成 3](#_Toc42264335)

[总结 3](#_Toc42264336)

[心得体会 3](#_Toc42264337)

# 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

* 数据准备之数据获取
* 数据准备之数据清洗
* 特征

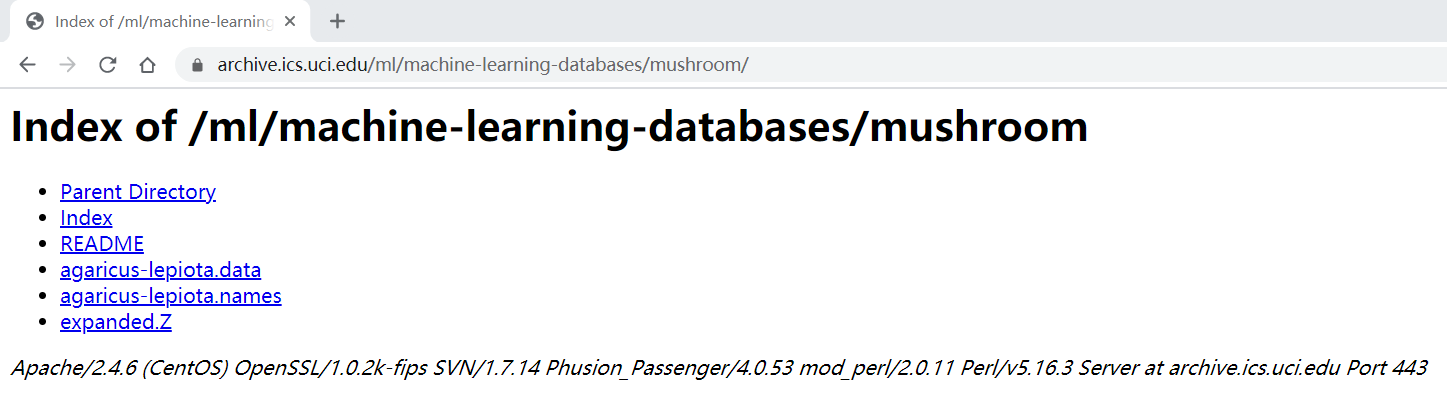
# 任务详细设计

包括详细的数据预处理过程、算法实现的设计和实验结果展示图，对实验结果的说明与描述等

## 问题定义

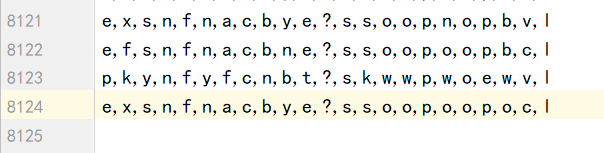
本实验数据来自于UCI的mushroom数据集，agaricus-lepiota.data为数据与标签，

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。



* 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。

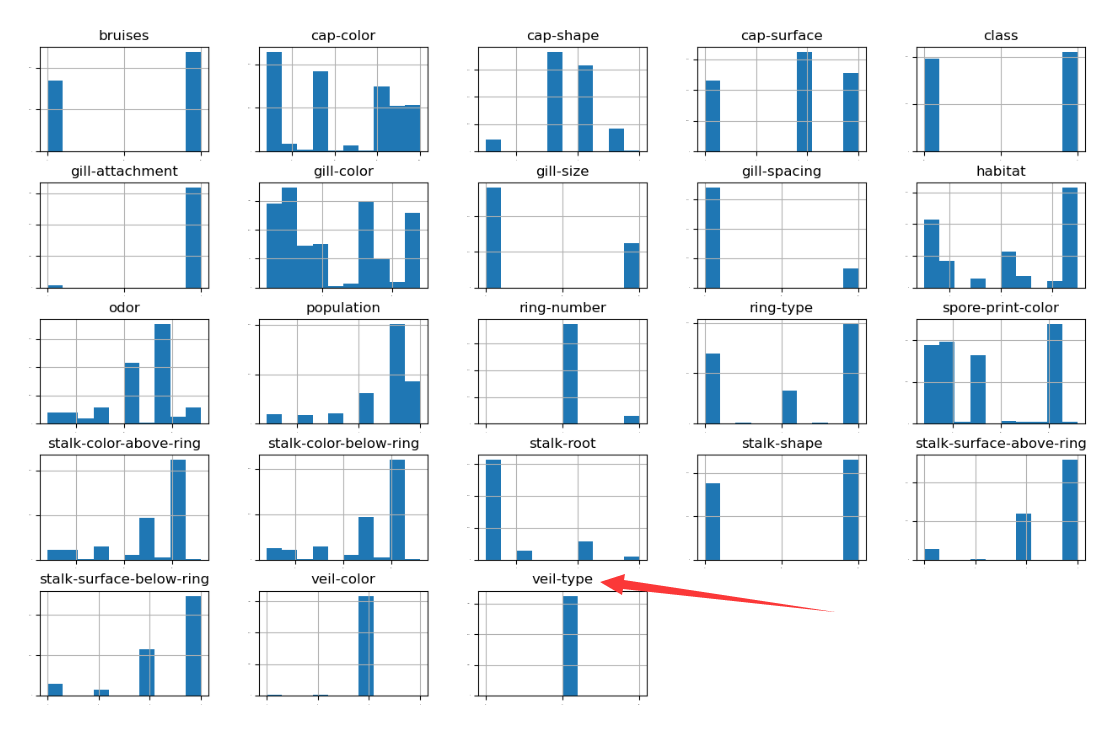
**def** \_\_char\_to\_int():  
 names = get\_names()  
 df = read\_csv(**'../../data/agaricus-lepiota.data'**, names=names)  
 *# 去掉缺失值多的一列* df.drop(**'****stalk-root'**, axis=1, inplace=**True**) *# print(df.shape)* dataSet = []  
 **for** d **in** df.\_values:  
 data = []  
 **for** cidx **in** range(len(d)):  
 *# 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

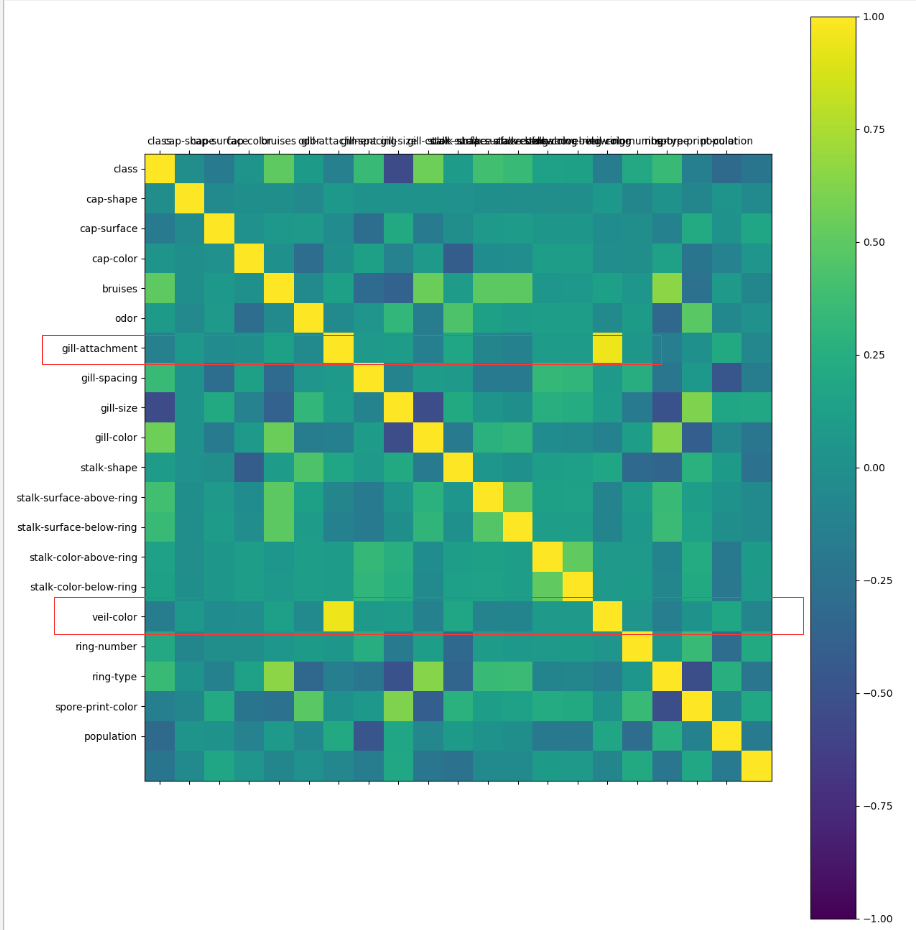
## 分析数据

* 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

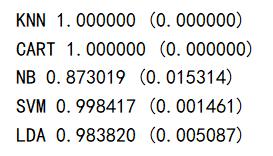
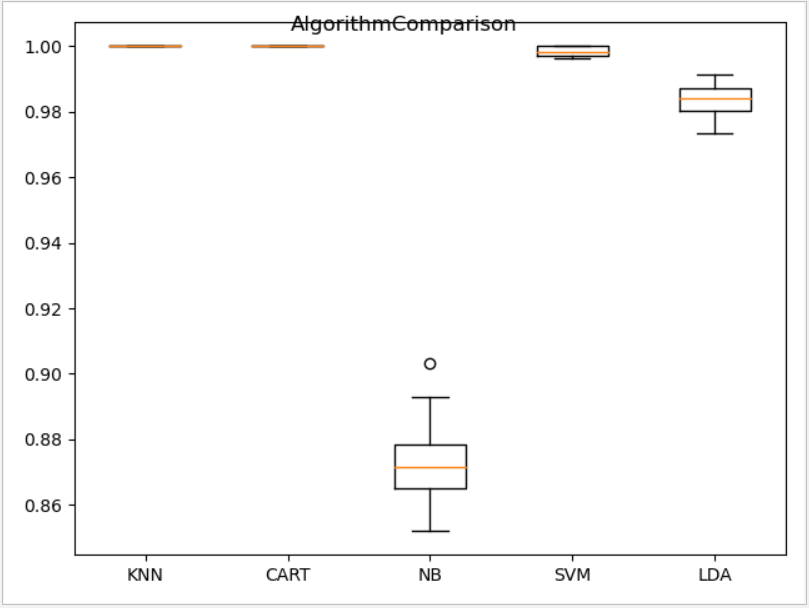
* 通过切分数据集将数据分为训练集以及测试集

**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%。

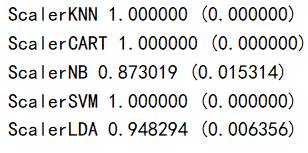
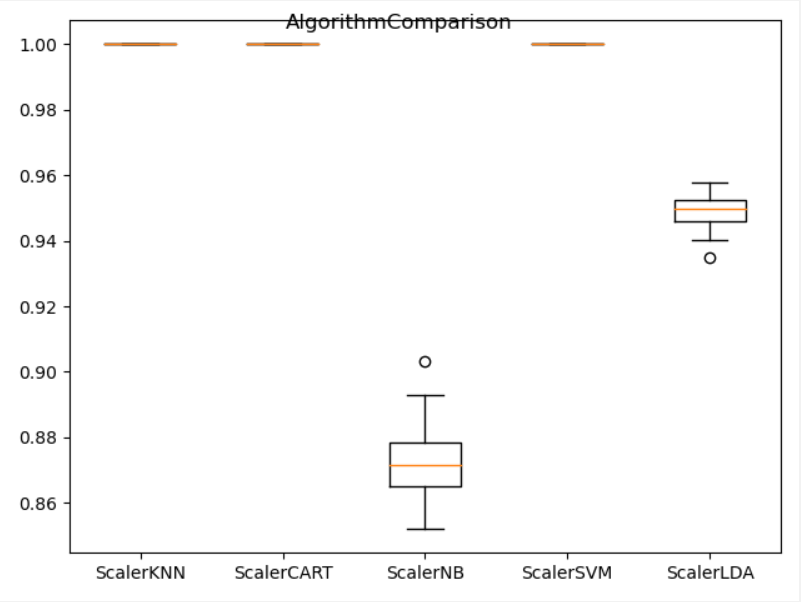
 

**from** sklearn.model\_selection **import** KFold, cross\_val\_score  
**from** sklearn.tree **import** DecisionTreeClassifier  
**from** sklearn.neighbors **import** KNeighborsClassifier  
**from** sklearn.naive\_bayes **import** GaussianNB  
**from** sklearn.linear\_model **import** LogisticRegression  
**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

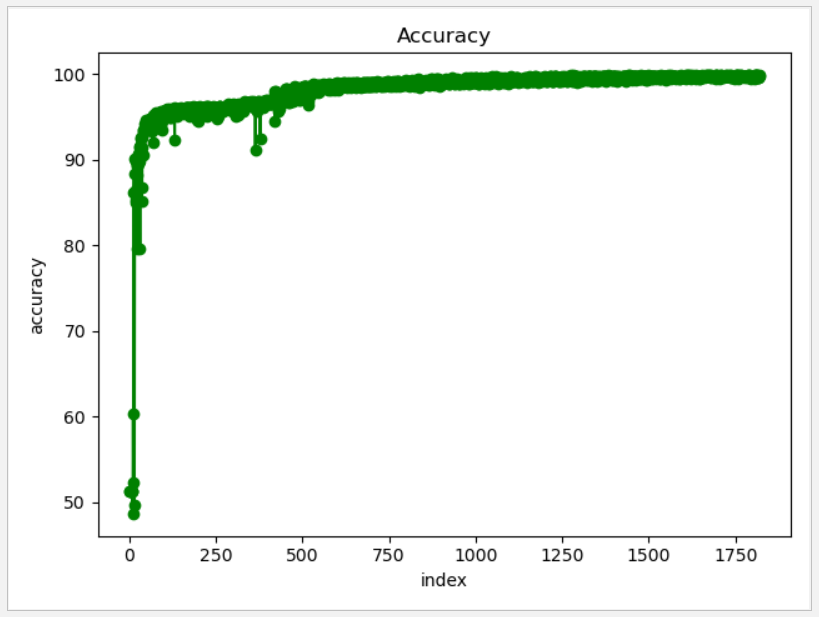
* 对上述模型进行优化，主要是使用Pipeline流水线对数据进行正态化处理。可以看到正态化数据之后，SVM模型的准确率也提升到了100%。

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

* 搭建包含一层隐含层的BP全连接神经网络分类模型



## 算法集成

## 总结

# 心得体会

## 李彬楷

## 陈臻

## 汤世展