数据挖掘可视化系统-需求文档 V1.0

作　　者：许继元 程络 黄倬熙 张平路
刘浩斌 邓志聪

学　　历：本科

指导老师 ：谢光强

目录

摘要

数据挖掘可视化系统（Data Mining Visualization System）通过数据挖掘理论、机器学习算法以及数据可视化等信息技术，并基于 Flask 框架搭建 Web 服务器，实现数据挖掘可视化。用户可以在线进入该数据挖掘平台，选择系统所提供的数据，观察数据的原始分布，并选择对应的机器学习算法进行数据挖掘，最后对数据挖掘结果进行可视化，使原本抽象的数据以简明的形式呈现出来，用户通过该数据挖掘系统可以达到对数据更深层的理解，了解数据之下更有价值的信息。

该项目基于 Python 等语言实现了一个**面向数据挖掘新手的无编程数据挖掘可视化系统**。以下阐述该项目一些功能的实现：提供原始数据的可视化分布、根据用户所选数据集提供相关机器学习算法进行数据挖掘、对数据挖掘结果进行可视化，为用户提供一个智能化、无编程化的数据挖掘可视化平台。

[关键词]：**数据挖掘、机器学习、数据可视化**

1. 需求分析

随着 5g 网络的兴建，国家高度重视人工智能的发展并专门为人工智能产业提出发展规划，人工智能领域将成为时代潮流的风口浪尖。对人工智能人才的缺口也会急速上升。数据挖掘作为人工智能的分支之一，也吸引了诸多学子。可是，抽象的算法和复杂的编程却让人却让人望而却步。

无论是想利用数据挖掘解决问题的编程小白，还是想要学习数据挖掘的菜鸟，复杂的 python 语言和抽象的机器学习算法都是横亘在他们面前的巨石。

对于本身不会编程但想要利用数据挖掘解决问题的人，无论是繁琐的编译器配置，以及漫长的 python 语言学习周期，都阻止他解决迫在眉睫的数据挖掘问题。

对于略有编程基础想要学习数据挖掘算法的人来说，即使有网上的课程，但缺乏可视化展示，依然会对数据挖掘学习产生阻碍。

目前市面还没有相应的工具，来解决此类的痛点。

本项目实现了数据挖掘的可视化功能，（支持用户自主上传数据集），提供多种数据集以及算法，以及简单方便的 UI 交互功能，让数据挖掘变得简单。

1. 支持多种数据集

本项目将提供多种数据集进行算法分析，如森林大火数据集，垃圾邮件数据集。

2. 多样化原始数据可视化方式

在原始数据中，多维数据十分常见，我们采用平行坐标图，雷达图，散点图等多种表现形式多元化，创新性的将原始数据可视化，使得用户能够直观的感受多维数据。

3. 结合多种数据挖掘算法

本项目将提供 SVM，多元线性回归，朴素贝叶斯，决策树等多种算法供用户进行选择，以不同方式挖掘出数据背后的规律，理解算法原理。

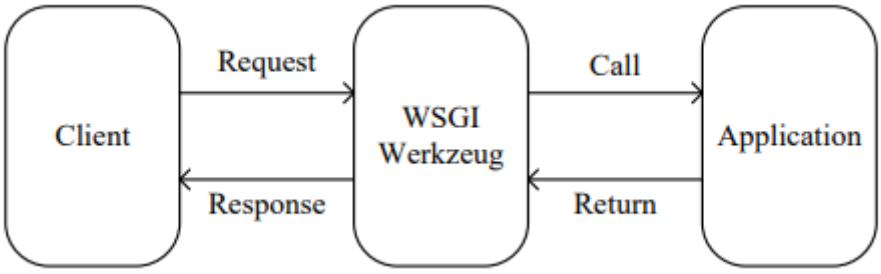
4. 个性化数据挖掘可视化

针对不同算法，我们将给出个性化的数据可视化方式，如决策树算法我们将以树状图的形式进行呈现，朴素贝叶斯，我们将使用词云图来达到可视化的目的

2. 实现技术概要

2.1 Flask 框架

Flask 是一个轻量级的可定制框架，使用 Python 语言编写，较其他同类型框架更为灵活、轻便、安全且容易上手。它可以很好地结合 MVC 模式进行开发，开发人员分工合作，小型团队在短时间内就可以完成功能丰富的中小型网站或 Web 服务的实现。另外，Flask 还有很强的定制性，用户可以根据自己的需求来添加相应的功能，在保持核心功能简单的同时实现功能的丰富与扩展，其强大的插件库可以让用户实现个性化的网站定制，开发出功能强大的网站。



Flask 的基本模式为在程序里将一个视图函数分配给一个 URL，每当用户访问这个 URL 时，系统就会执行给该 URL 分配好的视图函数，获取函数的返回值并将其显示到浏览

器上，其工作过程见图。

IT 运维的基本点为安全、稳定、高效，运维自动化的目的就是为了提高运维效率，Flask[9]开发快捷的特点正好符合运维的高效性需求。在项目迭代开发的过程中，所需要实现的运维功能以及扩展会逐渐增多，针对这一特点更是需要使用易扩展的 Flask 框架。另外，由于每个公司对运维的需求不同，所要实现的功能也必须有针对性地来设计，Flask 可以很好地完成这个任务。

2.2 数据挖掘分类算法

2.2.1 KNN 算法

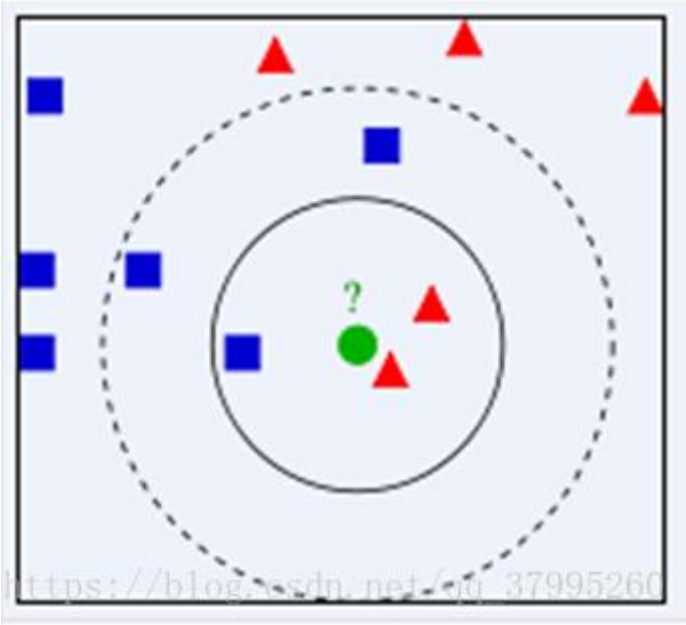
实现目的:

利用 KNN 算法对建立模型并对数据预测实现可视化

实现原理:

kNN 算法的核心思想是如果一个样本在特征空间中的 k 个最相邻的样本中的大多数属于某一个类别，则该样本也属于这个类别，并具有这个类别上样本的特性。该方法在确定分类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。

如下图，绿色圆要被决定赋予哪个类，是红色三角形还是蓝色四方形？如果 K=3，由于红色三角形所占比例为 2/3，绿色圆将被赋予红色三角形那个类，如果 K=5，由于蓝色四方形比例为 3/5，因此绿色圆被赋予蓝色四方形类。



实现过程：

1. 初始化训练集和类别；

2. 计算测试集样本与训练集样本的欧氏距离；

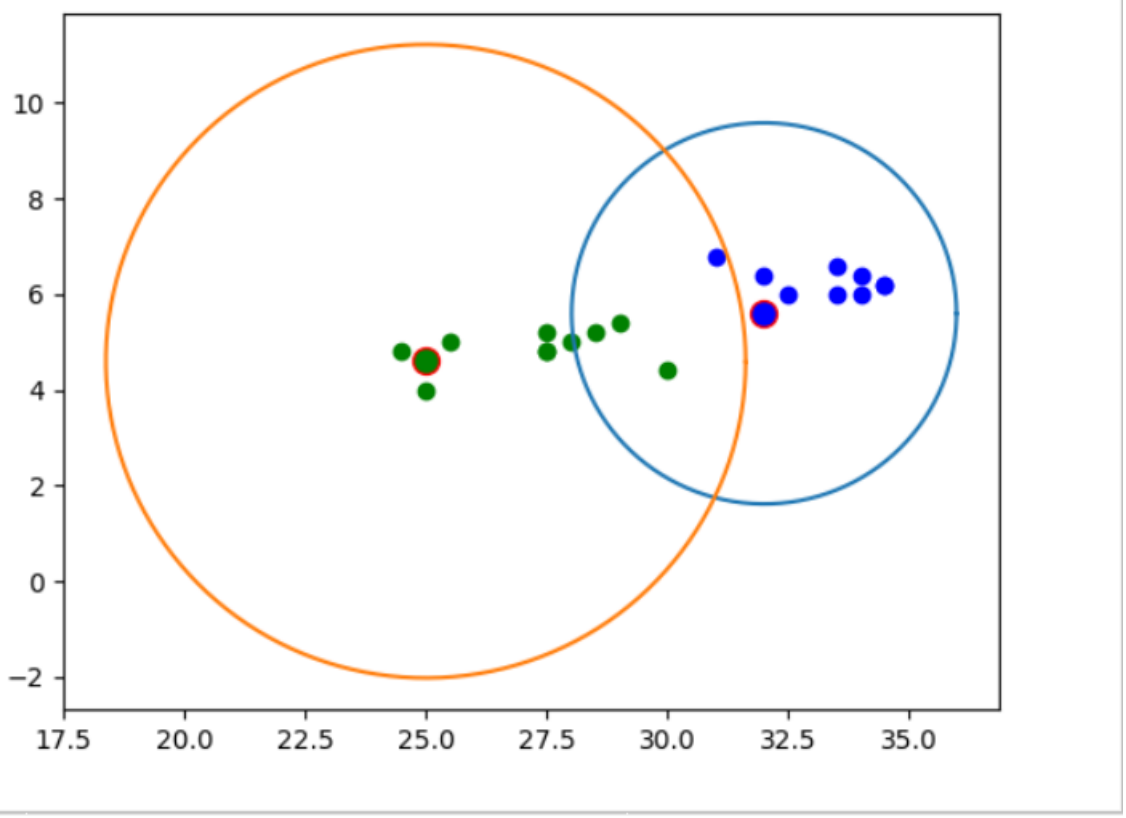
3. 根据欧氏距离大小对训练集样本进行升序排序；

4. 选取欧式距离最小的前 K 个训练样本，统计其在各类别中的频率；

5. 返回频率最大的类别，即测试集样本属于该类别。

实现效果：

使用户能够查看模型评估结果以及可视化结果。



2.2.2 朴素贝叶斯算法

实现目的:

实现对文本（如垃圾邮件）的词库建立，以及新文本预测，并实现算法可视化。

实现原理:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

P(A)是先验概率，表示每种类别分布的概率；

P(B|A)是条件概率，表示在某种类别前提下，某事发生的概率；该条件概率可通过统计而得出，这里需要引入极大似然估计概念，详见后文。

P(A|B)是后验概率，表示某事发生了，并且它属于某一类别的概率，有了这个后验概

率，便可对样本进行分类。后验概率越大，说明某事物属于这个类别的可能性越大，便越有理由把它归到这个类别下。

实现步骤：

1. x={a1,a2,....,am}x =\{a_1,a_2,....,a_m\}$x=\{a_1,a_2,....,a_m\}$为待分类项， 每个 a 为 x 的一个特征属性

2.有类别集合$C= {y_1,y_2,...,y_n} $

3.计算 P(y1|x),P(y2|x),...,P(yn|x)P(y_1|x),P(y_2|x),...,P(y_n|x)

如果 P(yk|x)=max{P(y1|x),P(y2|x),...,P(yn|x)}

实现结果：

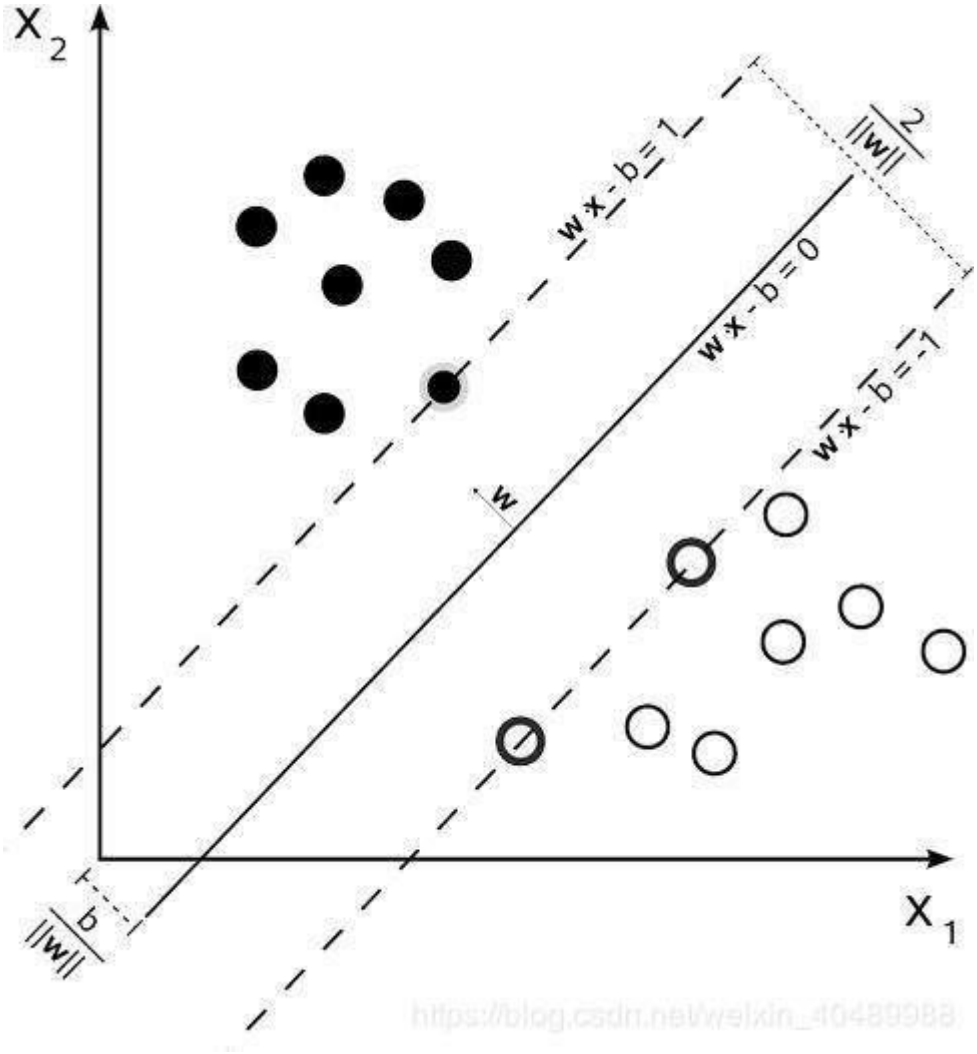
2.2.3 支持向量机算法

实现目的：

寻找到超平面并对数据进行分类，以及实现数据可视化。

实现原理：

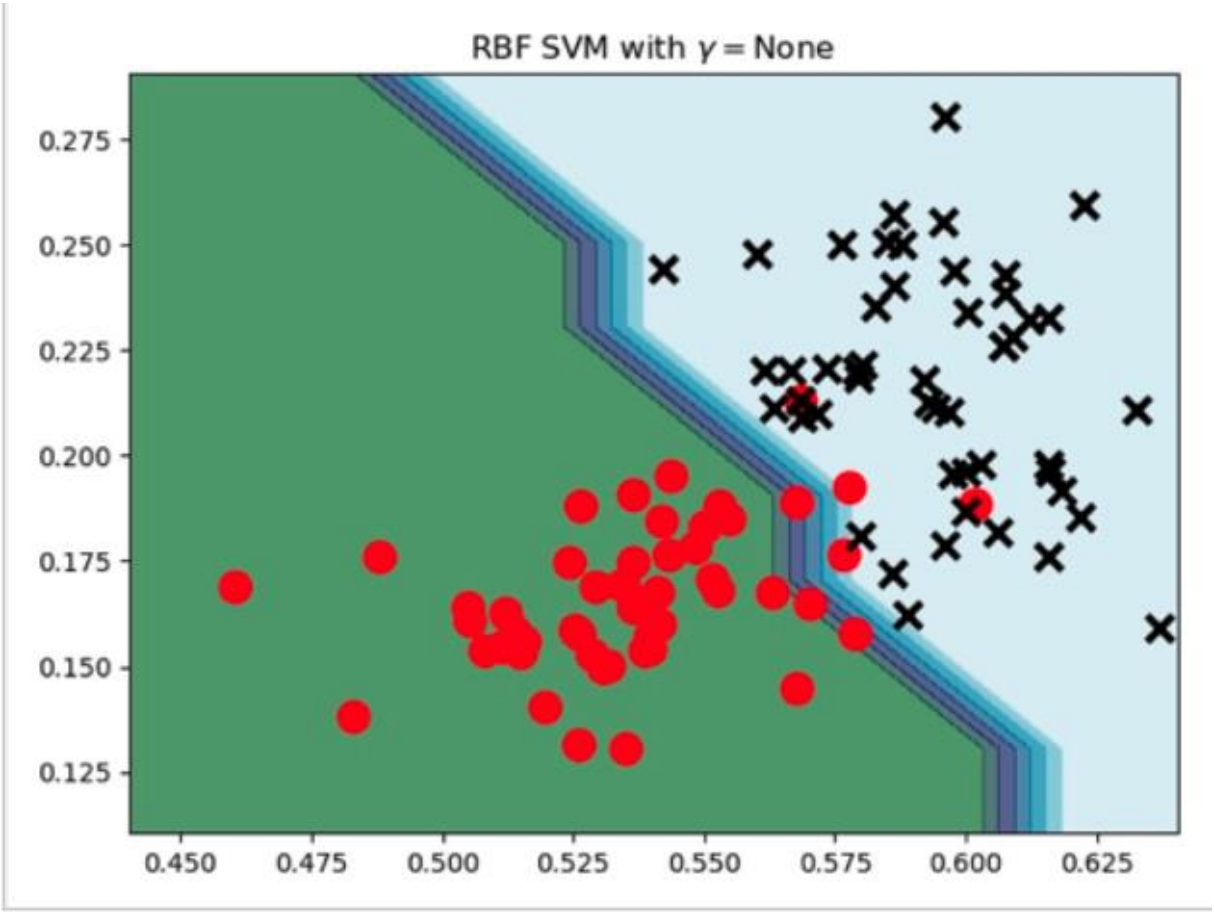
超平面和最近的数据点之间的间隔被称为分离边缘，用 P 表示。支持向量机的目标是找到一个特殊的超平面，对于这个超平面分离边缘 P 最大。在这个条件下，决策曲面称为最优超平面。



实现步骤：

1. 将寻找最优划分直线问题转化为凸函数优化求解。
2. 引入拉格朗日算子进行求导，将问题再次转换为另一个凸函数求解。
3. 引入松弛变量，通过 SMO 算法，利用多次迭代得到最佳的结果。

实现结果：



2.3 数据挖掘预测算法

2.3.1 多元线性回归

实现目的：

求出多元线性回归的回归方程，对数据进行预测并实现可视化

实现原理：

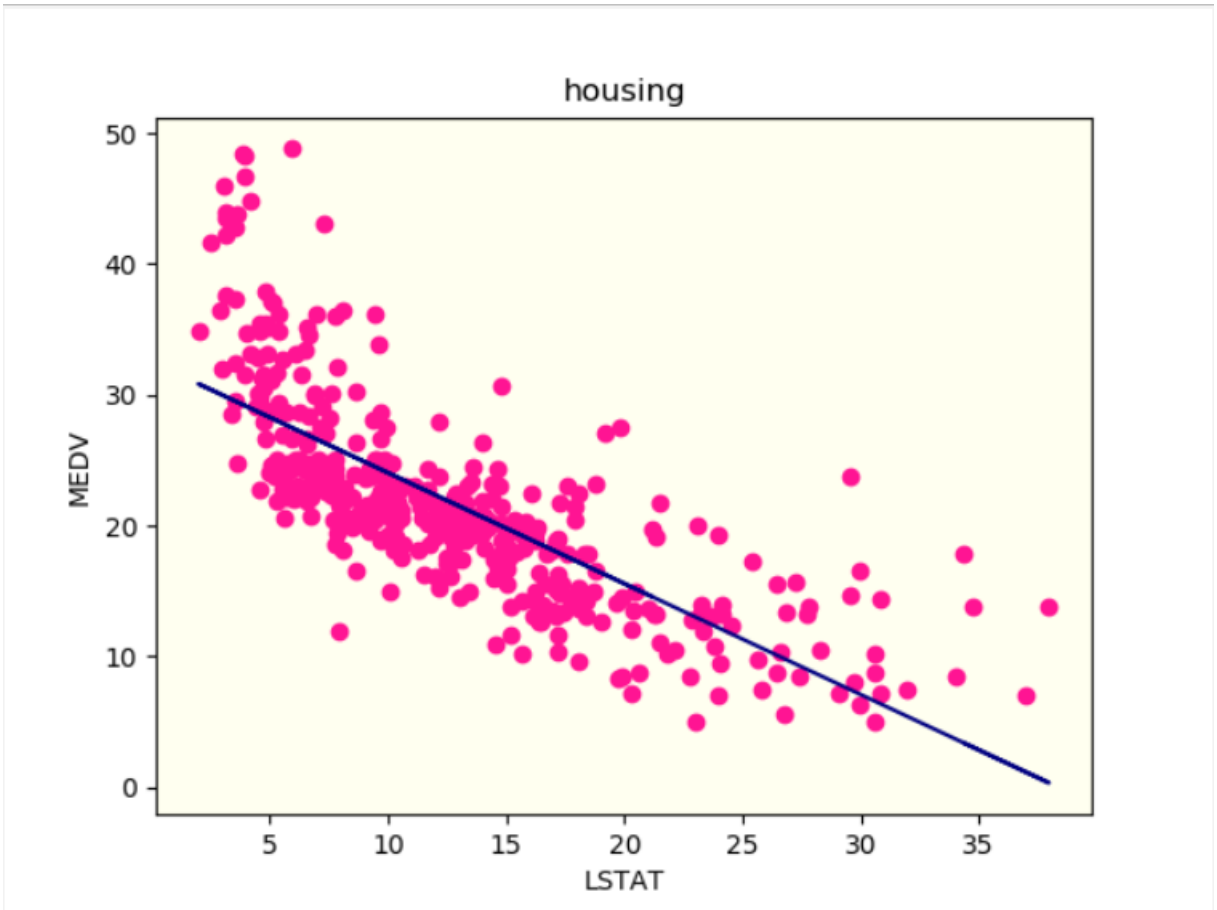
在统计学中，线性回归方程是利用最小二乘函数对一个或多个自变量之间关系进行建模的一种回归分析。这种函数是一个或多个称为回归系数的模型参数的线

性组合。只有一个自变量的情况称为简单回归，大于一个自变量的情况叫多元回归。

实现步骤:

1. 将数据矩阵展为增广矩阵
2. 利用最小二乘法求出权重
3. 带入测试集进行检验

实现结果:



2.3.2 逻辑回归

实现目的:

建立逻辑回归的模型，求得回归方程，实现数据的预测以及可视化。

实现原理:

逻辑回归假设数据服从伯努利分布，通过极大化似然函数的方法，运用梯度下降来求解参数，来达到将数据二分类的目的。

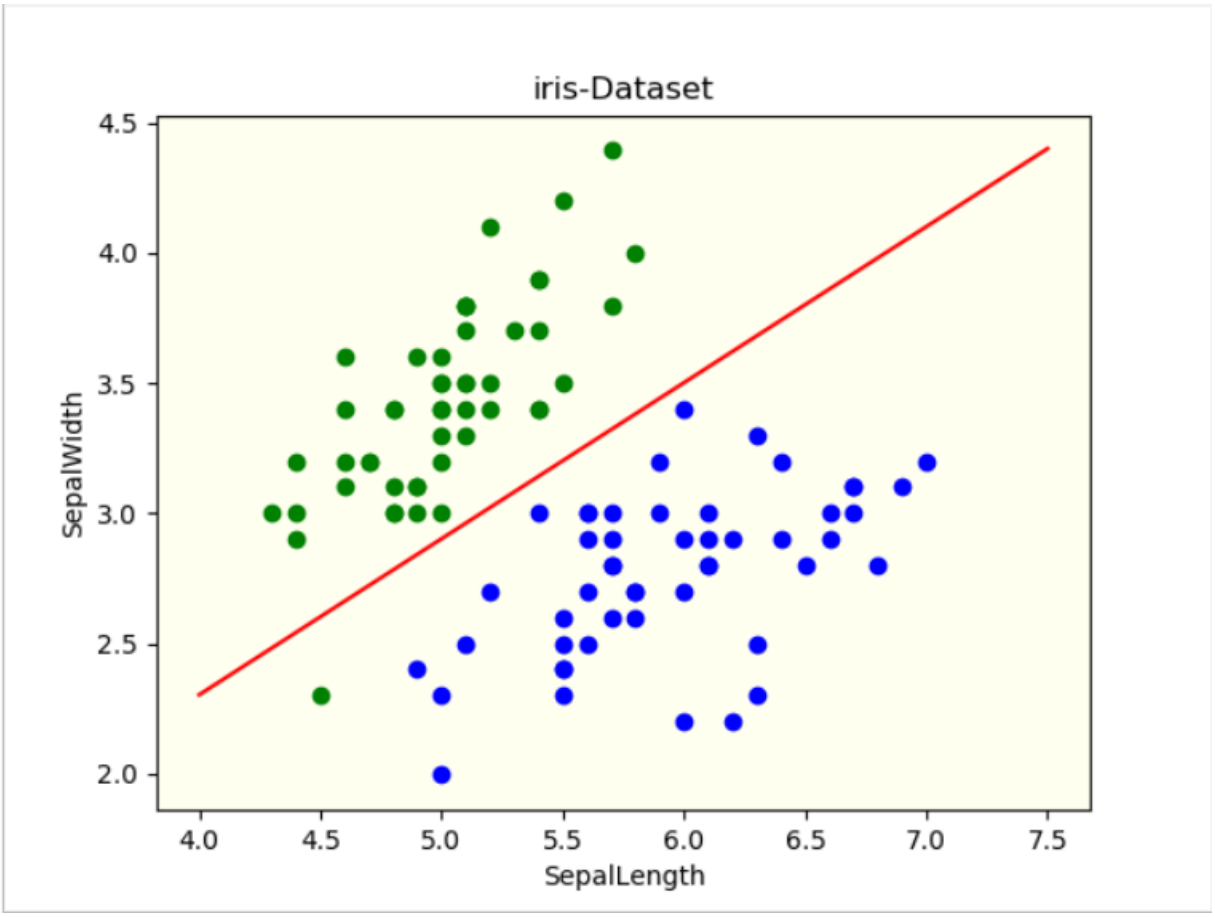
实现过程:

1.寻找一个合适的预测函数,一般是 h 函数(即 hypothesis 函数)。这个函数就是我们要找分类函数

2.构造一个 Cost 函数(即损失函数)，该函数用来表示预测函数（h）与训练数据类别（y）之间的偏差。

3.多次迭代求得最佳的权重，建立最合适的回归模型。

实现结果:



2.3.3 决策树

实现目的：

找到决策树的决策过程，并实现数据的可视化。

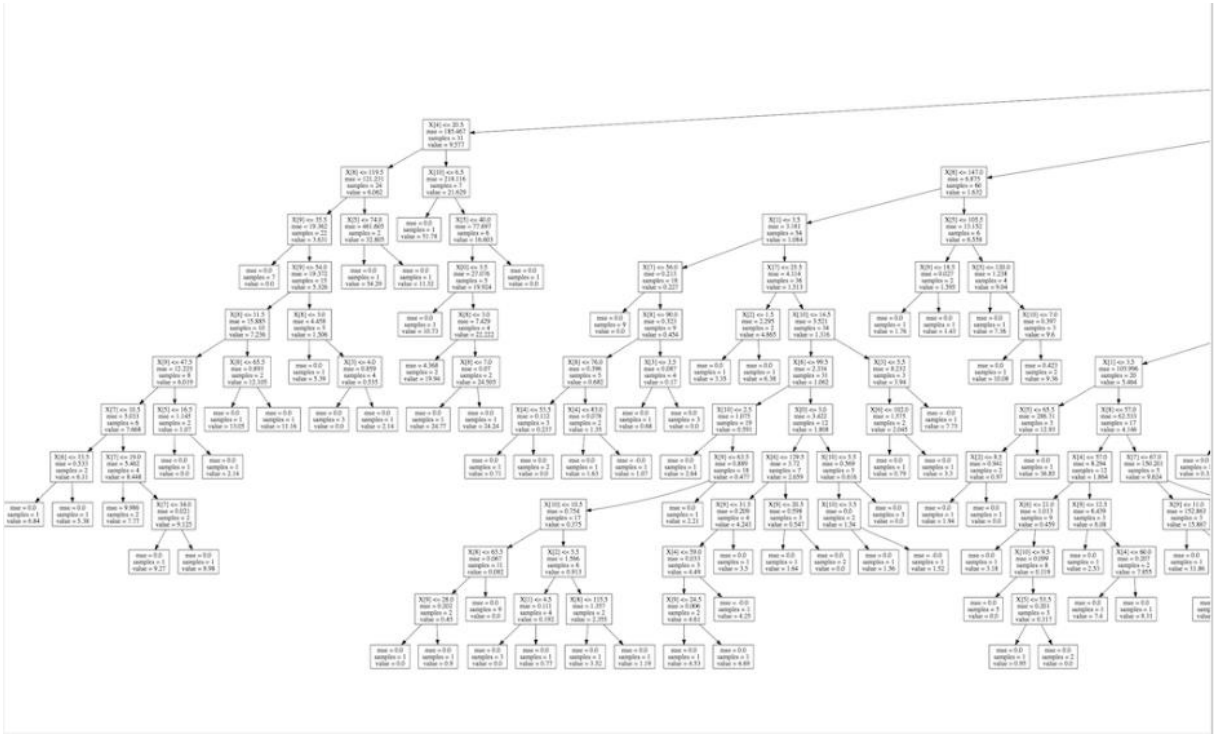
实现原理：

决策树（Decision Tree）是在已知各种情况发生概率的情况下，通过构成决策树来求取净现值的期望值大于 0 的概率，是直观运用概率分析的一种图解法。通俗的讲，决策树就是带有特殊含义的数据结构中的树结构，其每个根结点（非叶子结点）代表数据的特征标签，根据该特征不同的特征值将数据划分成几个子集，每个子集都是这个根结点的子树，然后对每个子树递归划分下去，而决策树的每个叶子结点则是数据的最终类别标签。

实现步骤：

1. 找到基尼系数最小的特征划分
2. 去除已经划分的特征，迭代步骤一。
3. 不断递归，开枝散叶生成决策树。

实现结果：



2.4 多维数据可视化

一般来说，数据挖掘算法能够处理的数据都是多维数据，但人只能看到并且理解三维以下的图像，因此多维数据可视化成为一个难题，也是数据挖掘可视化系统的重中之重。

2.4.1Echarts 插件

ECharts，一个使用 JavaScript 实现的开源可视化库，可以流畅的运行在 PC 和移动设备上，兼容当前绝大部分浏览器（IE8/9/10/11，Chrome，Firefox，Safari 等），底层依赖轻量级的矢量图形库 ZRender，提供直观，交互丰富，可高度个性化定制的数据可视化图表。

ECharts 提供了常规的折线图、柱状图、散点图、饼图、K 线图，用于统计的盒形图，用于地理数据可视化的地图、热力图、线图，用于关系数据可视化的关系图、旭日图，多维数据可视化的平行坐标，还有用于 BI 的漏斗图，仪表盘，并且支持图与图之间的混搭。

2018 年 3 月全球著名开源社区 Apache 宣布百度 ECharts 进入 Apache 孵化器。

优点：

丰富的图表类型

ECharts 提供了常规的折线图、柱状图、散点图、饼图、k 线图，用于统计的盒形图，用于地理数据可视化的地图、热力图、线图，用于数据关系可视化的关系图，treemap，多维数据可视化的平行坐标，还有用于 BI 的漏斗图，仪表盘，并且支持图与图之间的混搭

动态数据

ECharts 由数据驱动，数据的改变驱动图表展现的改变。因此动态数据的实现也变得异常简单，只需获取数据，填入数据，ECharts 会找到两组数据之间的差异然后通过合适的动画去表现数据的变化。

强大的动画支持

提供 promise 式的动画接口和常用缓动函数，轻松实现各种动画需求

易于扩展

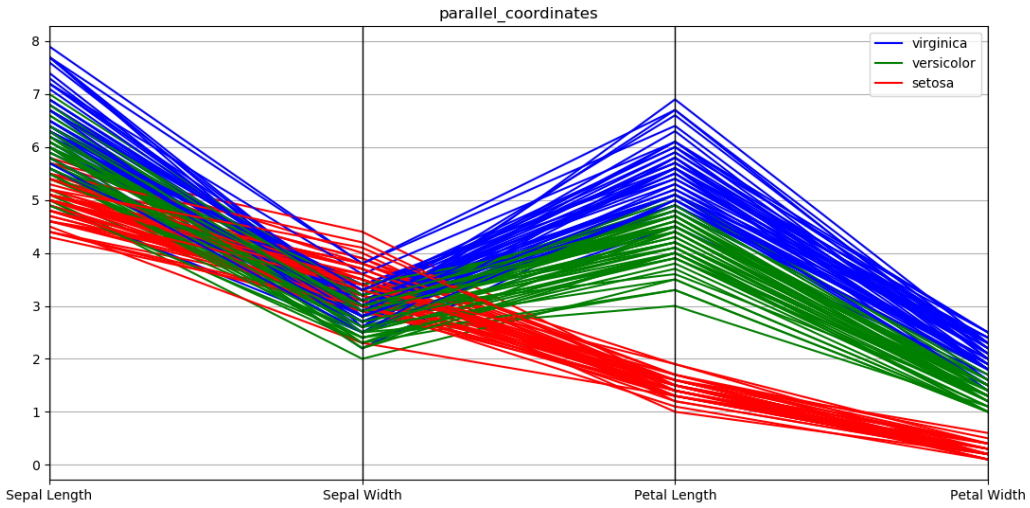
分而治之的图形定义策略允许你扩展自己独有的图形元素，你既可以完整实现三个接口方法（brush,drift,isCover），也可以通过 base 派生后仅实现需要关心的图形细节。

2.4.2 pandas 的 plotting 函数

Python 中的 pandas 中 plotting 可以将多维数据可视化，并提供了四种方法：

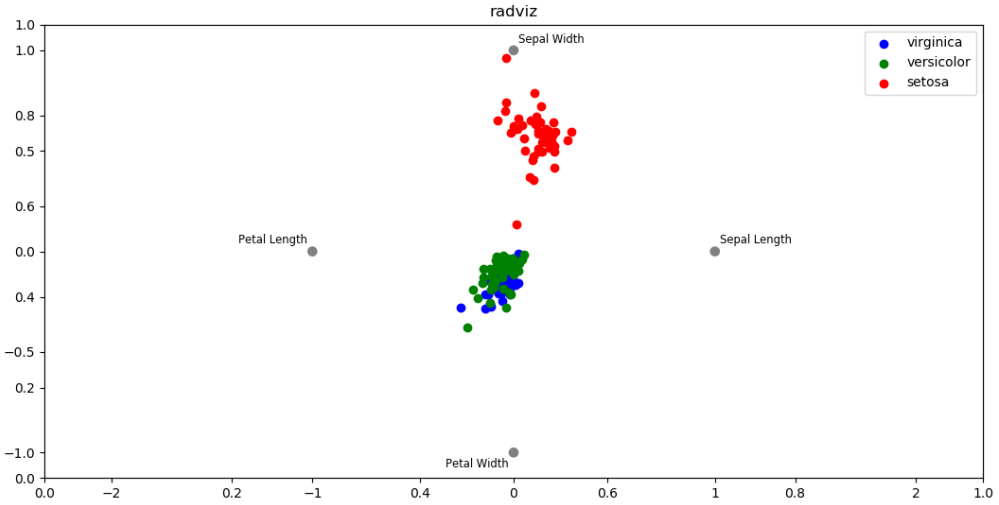
1. 平行坐标:

图中每条垂直的线代表一个特征，表中一行的数据在图中表现为一条折线，不同颜色的线表示不同的类别。



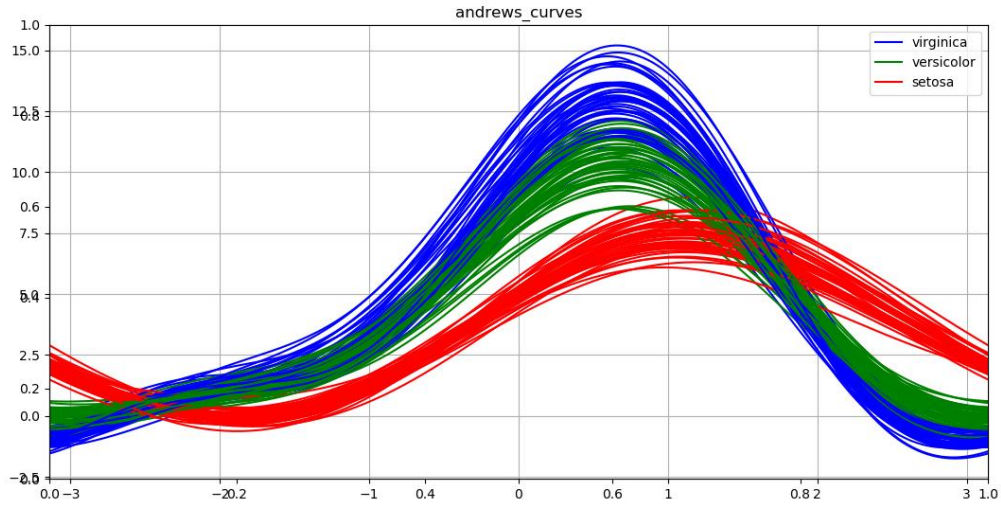
2. RadViz 雷达图:

4 个特征对应于单位圆上的 4 个点，圆中每一个散点代表表中一行数据。可以想象为每个散点上都有 4 条线分别连接到 4 个特征点上，而特征值（经过标准化处理）就表示这 4 条线施加在散点上的力，每个点的位置恰好使其受力平衡。



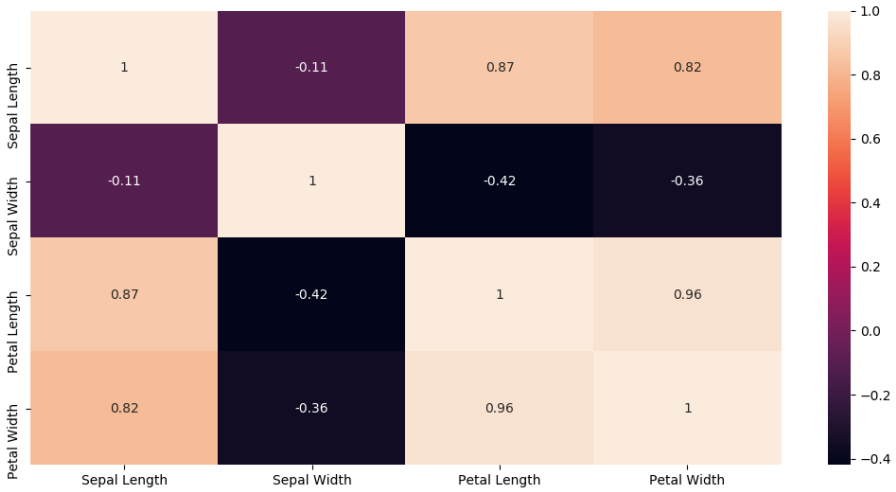
3. Andrew 曲线

特征值转化为傅里叶序列的系数，不同颜色的曲线代表不同的类别。



4.相关系数热力图

表示不同特征之间的相关性（Pearson 相关系数），数值越大，相关性越高。



2.5 单一进程下的多用户调度系统

哈希表是根据设定的哈希函数 H（key）和处理冲突方法将一组关键字映射到一个有限的地址区间上，并以关键字在地址区间中的象作为记录在表中的存储位置，这种表称为哈希表或散列，所得存储位置称为哈希地址或散列地址。作为线性数据结构与表格和队列等相比，哈希表无疑是查找速度比较快的一种。

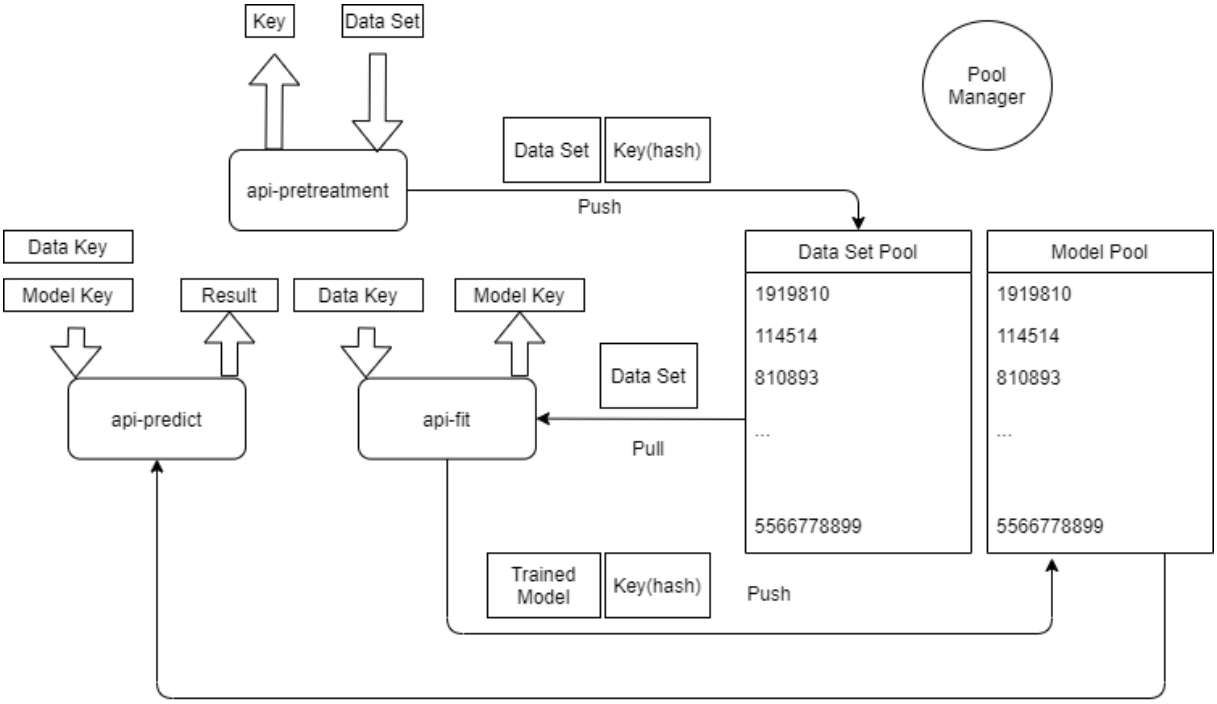
通过将单向数学函数（有时称为“哈希算法”）应用到任意数量的数据所得到的固定大小的结果。如果输入数据中有变化，则哈希也会发生变化。哈希可用于许多操作，包括身份验证和数字签名。也称为“消息摘要”。

简单解释：哈希（Hash）算法，即散列函数。它是一种单向密码体制，即它是一个从明文到密文的不可逆的映射，只有加密过程，没有解密过程。同时，哈希函数可

以将任意长度的输入经过变化以后得到固定长度的输出。哈希函数的这种单向特征和输出数据长度固定的特征使得它可以生成消息或者数据。

hash 编码目的：

我们在调度系统中运用 hash 编码，将每一种算法和训练集，对应成独一无二的 hash key，从而大大加快了调度系统的效率。



调度系统的逻辑模型-图 3.5.1

系统显然不会只受到单一用户的访问，因此在面对多用户单一进程的并发处理，我们需要确保每一个用户只能访问到自己的数据集以及训练模型，以最大程度避免数据泄漏的安全问题。

为此我们需要通过一个线性的进程去完成对于多用户的并发处理形式，因此引入了 hashKey 作为每个用户请求数据集以及模型的唯一密钥，最大程度保障用户隐私安全，同时也是调用后台中的 C/C++库进行数据的处理，加快数据处理速度。简单工作流程如图 3.5.1 所示：首先用户提交对数据进行预处理（包括去除 NaN、独热编码等）的请求，将数据集以及其他参数提交，服务器为用户返回一个独一无二的 hashKey 作为密钥，以最大程度杜绝数据劫持的发生。同时用户借助这个密钥以及选好的算法上传到服务器进行训练，训练完毕之后得到完成训练的模型的 hashKey。当用户需要进行预测的时候，只需要将数据集以及模型的密钥进行提交，再由服务器返回预测的结果并

显示在前端上，仅两次的实际有效的单向数据传输一方面降低了数据的吞吐量，另一方面也降低了数据被劫持的概率。

3. 系统实现

3.1 数据挖掘分类算法

3.1.1 KNN 算法

伪代码:

复杂来写：

```
    def fit(train, k):
   self.train = train
   self.k = k
def predict(test):
   result = []
   for x in test:
     # a. 从训练数据 train 中获取和当前数据 x 距离最近的 k 个样本
     neighbors = fetch_k_neighbors(self.train, x, self.k)

     # b. 合并这 K 个最近样本，得到预测值
  # b1. 统计一下各个类别 label 出现的次数
  label_2_count_dict = {}
  for neighbor in neighbors:
    # b11. 获取当前样本 neighbor 的标签值
    label = neighbor.label
```

```
    # b12. 将这个 label 添加到字典中
    if label not in label_2_count_dict:
      label_2_count_dict[label] = 1
    else:
      label_2_count_dict[label] += 1
  # b2. 从这个字典中获取出现次数最多的 label 标签值作为预测值
  max_label_count = 0
  max_label = None
  for label in label_2_count_dict:
       # 获取当前 label 对应出现的 count 数量
    count = label_2_count_dict[label]
    # 将当前 count 和最大值进行比较，选择/保留最大的 count
    if count > max_label_count:
      max_label_count = count
   max_label = label
  # b3. 将预测值添加到集合中
  result.append(max_label)
   return result
```

3.1.2 朴素贝叶斯算法

伪代码:

```
def spamTest():
    docList = []
    classList = []
```

```
    fullText = []
    for i in range(1, 26):  # 遍历 25 个 txt 文件
        wordList = textParse(
            open('垃圾邮箱/spam/%d.txt' % i, 'r').read())  # 读取每个垃圾邮件，并字符串转换成字符串列表

        docList.append(wordList)
        fullText.append(wordList)
        classList.append(1)  # 标记垃圾邮件，1 表示垃圾文件
        wordList = textParse(
            open('垃圾邮箱/ham/%d.txt' % i, 'r').read())  # 读取每个非垃圾邮件，并字符串转换成字符串列表
        docList.append(wordList)
        fullText.append(wordList)
        classList.append(0)  # 标记正常邮件，0 表示正常文件

    vocabList = createVocabList(docList)  # 创建词汇表，不重复

    trainingSet = list(range(50))
    testSet = []  # 创建存储训练集的索引值的列表和测试集的索引值的列表
    for i in range(10):  # 从 50 个邮件中，随机挑选出 40 个作为训练集,10 个做测试集
        randIndex = int(random.uniform(0, len(trainingSet)))  # 随机选取索索引值
        testSet.append(trainingSet[randIndex])  # 添加测试集的索引值
```

```
        del (trainingSet[randIndex])  # 在训练集列表中删除添加到测试集的索引值
    trainMat = []
    trainClasses = []  # 创建训练集矩阵和训练集类别标签系向量

    for docIndex in trainingSet:  # 遍历训练集
        trainMat.append(setOfWords2Vec(vocabList, docList[docIndex]))  # 将生成的词集模型添加到训练矩阵中

        trainClasses.append(classList[docIndex])  # 将类别添加到训练集类别标签系向量中

    p0V, p1V, pSpam = trainNB0(np.array(trainMat), np.array(trainClasses))
# 训练朴素贝叶斯模型
    errorCount = 0  # 错误分类计数
    for docIndex in testSet:  # 遍历测试集
        wordVector = setOfWords2Vec(vocabList, docList[docIndex])  # 测试集的词集模型
        if classifyNB(np.array(wordVector), p0V, p1V, pSpam) !=
classList[docIndex]:  # 如果分类错误
            errorCount += 1  # 错误计数加1
            print("分类错误的测试集：", docList[docIndex])
    print('错误率：%.2f%%' % (float(errorCount) / len(testSet) * 100))
```

3.1.3 支持向量机算法

伪代码：

```
alphaChanged = 0
for i in range(m):
    #步骤1：计算误差
    fxi =
float(np.multiply(alpha,labelMatrix).T*(dataMatrix*dataMatrix[i,:].T)) +b
    Ei = fxi - float(labelMatrix[i])

    if ((labelMatrix[i] * Ei < -toler) and (alpha[i] < C)) or
((labelMatrix[i] * Ei > toler) and (alpha[i] > 0)):
        j = selectJrand(i,m)
        #计算误差Ej
        fxj =
float(np.multiply(alpha,labelMatrix).T*(dataMatrix*dataMatrix[j,:].T)) +b
        Ej = fxj - float(labelMatrix[j])
        #保存下更新前的alphas
        alphaIold = alpha[i].copy()
        alphaJold = alpha[j].copy()
        #使用深拷贝

        #分情况讨论
        if(labelMatrix[i] != labelMatrix[j]):
            L = max(0,alpha[j]-alpha[i])
            H = min(C,C+alpha[j]-alpha[i])
        else:
            L = max(0,alpha[j]+alpha[i]-C)
```

```
            H = min(C,alpha[j]+alpha[i])
        if L==H: continue
        #步骤 3，计算学习率 eta
        eta = 2.0 * dataMatrix[i,:]*dataMatrix[j,:].T -
dataMatrix[i,:]*dataMatrix[i,:].T - dataMatrix[j,:]*dataMatrix[j,:].T
        if eta >= 0:continue
        #步骤 4：更新 alpha——j
        alpha[j] -= labelMatrix[j]*(Ei - Ej)/eta
        #步骤 6：修剪 alpha—j
        alpha[j] = clipAlpha(alpha[j],H,L)
        if(abs(alpha[j]-alphaJold<0.00001)):continue
        #步骤 6：更新 alpha——i
        alpha[i] += labelMatrix[j]*labelMatrix[i]*(alphaJold - alpha[j])
        #步骤 7：更新 b_1 和 b_2
        b1 = b - Ei - labelMatrix[i] * (alpha[i] - alphaIold) *
dataMatrix[i, :] * dataMatrix[i, :].T - labelMatrix[j] * (alpha[j] -
alphaJold) * dataMatrix[i, :] * dataMatrix[j, :].T
        b2 = b - Ej - labelMatrix[i] * (alpha[i] - alphaIold) *
dataMatrix[i, :] * dataMatrix[j, :].T - labelMatrix[j] * (alpha[j] -
alphaJold) * dataMatrix[j, :] * dataMatrix[j, :].T
        # 步骤 8：根据 b_1 和 b_2 更新 b
        if (0 < alpha[i]) and (C > alpha[i]):
            b = b1
        elif (0 < alpha[j]) and (C > alpha[j]):
            b = b2
```

```
            else:
                b = (b1 + b2) / 2.0
            # 统计优化次数
            alphaChanged += 1
            # 打印统计信息
            print("第%d 次迭代 样本:%d, alpha 优化次数:%d" % (iter_num, i,
alphaChanged))
    if (alphaChanged == 0):
        iter_num += 1
    else:
        iter_num = 0
    print("迭代次数: %d" % iter_num)
```

3.2 数据挖掘预测算法

3.2.1 多元线性回归

伪代码:

```
X_b=np.hstack([np.ones((len(X_train),1)),X_train])#增广矩阵
self._theta=np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y_train)#计算 theta
self.coef_=self._theta[1:]
self.interception_=self._theta[0]
```

3.2.2 逻辑回归

伪代码:

```
while True:
```

```
    oldJ = self.costFunc(Xb, y)#the old cost value
    # 注意预测函数中使用的参数是未更新的
    c = sigmodFormatrix(Xb, self.thetas) - y
    for j in range(dimension):
        self.thetas[j] = self.thetas[j] - alpha * np.sum(c * Xb[:,
j])#update theta
    newJ = self.costFunc(Xb, y)
    if newJ == oldJ or math.fabs(newJ - oldJ) < accuracy:
        print("代价函数迭代到最小值，退出！")
        print("收敛到:", newJ)
        break
    print("迭代第", count, "次!")
    print("代价函数上一次的差:", (newJ - oldJ))
    count += 1
```

3.2.3 决策树

伪代码:

```
double Empirical_entropy(Node *p){  return 当前数据集的经验熵 HD;     }
double Conditional_entropy(int attr_index,Node *p){
 return 索引位置为attr_index的特征的条件熵HDA ;
}
int ID3(Node *p){
 double HD=Empirical_entropy(p);//经验熵
 double HDA[10];//条件熵
```

```
 for i=0:p->attrsize:
  HDA[i]=Conditional_entropy(i,p);
  G[i]=HD-HDA[i]//信息增益
 return 信息增益最大的特征;
}
double SplitInfo(int attr_index,Node* p){
 return 索引位置为attr_index的特征的熵 ;
}
int C4_5(Node* p){
 double HD=Empirical_entropy(p);
 for i=0:p->attrsize:
  HDA=Conditional_entropy(i,p);//条件熵
  double sp_info=SplitInfo(i,p);//特征的熵
  gainRatio[i]=(HD-HDA)/sp_info;//信息增益率
 return 信息增益率最大的特征;
}
double gini(int attr_index,Node* p){
 return 索引位置为attr_index的特征的gini系数 ;
}
int CART(Node* p){
 double Gini[10];
 for i=0:p->attrsize:
  Gini[i]=gini(i,p);
 return 基尼系数最小的特征;
```

```
}
int choose_attr(Node *p){
 int attr;//被选中的特征，下面有三种选择方法
 attr=ID3(p);
 //attr=C4_5(p);
 //attr=CART(p);
 return attr;
}
void decide_final_label(Node* p){
 多数投票，在数据集中     正标签多 return 1;      负标签多 return -1;    }
bool meet_with_bound(Node* p){
 if 符合边界条件一：所有数据集样本的标签都相同; return 1;
 if 边界条件二：所有数据集的每个特征的取值都相同; return 1;
 else return 0;//即继续向下分支
}
void recursive(Node *p){
 if(meet_with_bound(p)){
  decide_final_label(p);//决定这个叶子结点的标签
  return;
 }
 未达到边界，则继续向下分支
 int attr_chosen=choose_attr(p);//选择一个当前特征集中最优代表性的特征
 int index_chosen;
 找出这个被选中的特征在特征集里的索引位置即 index_chosen;
```

int maxnum,minnum;

分别求出特征取值的最大最小值；

for i=minnum:maxnum :

Node *tem=new Node;

新节点的特征集是父节点特征集除去被选中的哪一个特征后所剩的特征集；

新节点的数据集是父节点被选中特征的取值为 i 的那些数据样本的集合（删掉被选中特征那一列）

if 被选中特征确实有 i 这个取值

将 tem 与父节点连接起来

else 释放 tem 节点

for i=0:子节点数目：

recursive(p->children[i]);

}

void validate(int index,Node *p){

验证集使用决策树验证 :

if 到达叶子节点 or 当前节点被剪枝

取得其标签作为该验证集样例的预测结果存储进数组 vali_label_result[]中；

return ;

for i=0:children.size():

找到了对应当前验证集文本特征值的节点，

则进入这个子节点继续往下遍历

validate(index,p->children[i]);

if 上面没有找到对应的特征值子节点

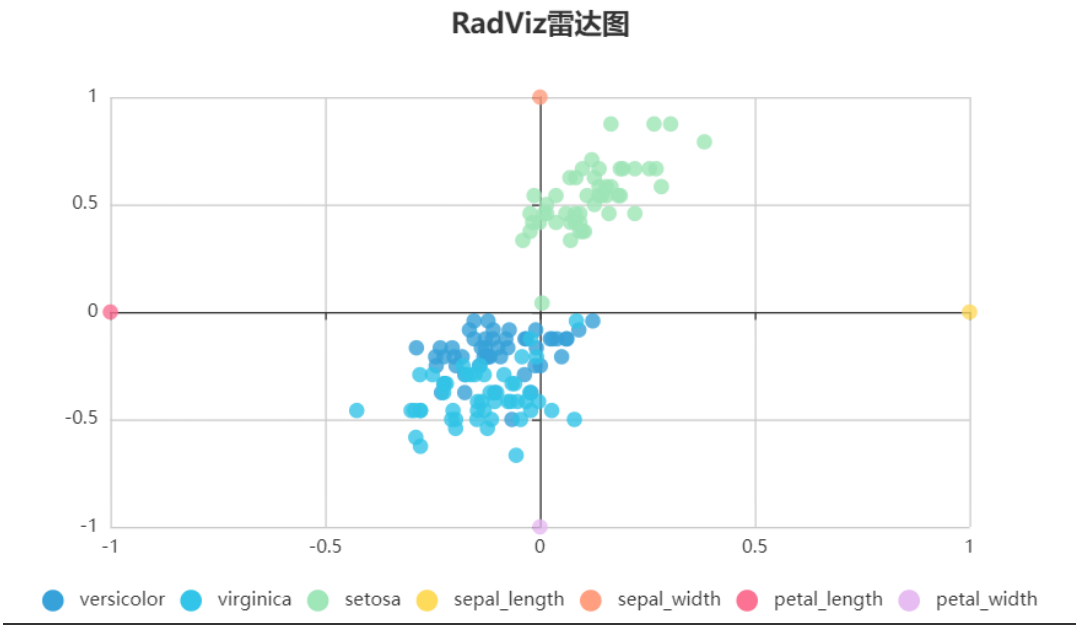
说明验证集出现训练集没有的新情况

```
  多少表决去标签存放数组 vali_label_result[]中；
}
void prune(Node* p){
 if 到达叶节点 return;
 for i=0:children.size():
  prune(p->children[i]);//后序遍历
  p->prune_or_not=1;
  decide_final_label(p);
  调用 validate()函数，验证集使用剪枝后的决策树预测
  根据数组 vali_label_result[]和正确答案数组 valilabel[]对比计算准确率
  if 准确率没有降低
   剪枝
  else p->prune_or_not=0;//不剪枝
}
void initial(){
 Readtext();
 初始化根节点：初始化数据集、特征集等等参数
}
void Readtext(){
 读取 train.csv 文件，每五个样本取前四个进训练集数组，
 取第五个进验证集数组。且分别用一维数组记录其标签
}
void test_recursive(int index,Node* p){
 用决策树预测测试集标签；
```

```
 与 validate 函数类似，只是预测结果存放数组不同，不作赘述；
}
void output_test_result(){文件流输出测试集预测结果}
void Readtest(){读取测试集文本进测试集数组}
int main(){
 initial(); //初始化根节点
 recursive(root);//开始递归建立决策树
 for(int i=0;i<valicnt;i++)//未剪枝前，验证集使用决策树预测
  validate(i,root);
 计算验证集预测结果准确率
 prune(root);//剪枝
 Readtest();
 for(int i=0;i<testcnt;i++)//决策树预测测试集
  test_recursive(i,root);
 output_test_result();//输出结果到 txt 文件
}
```

3.3 多维数据可视化

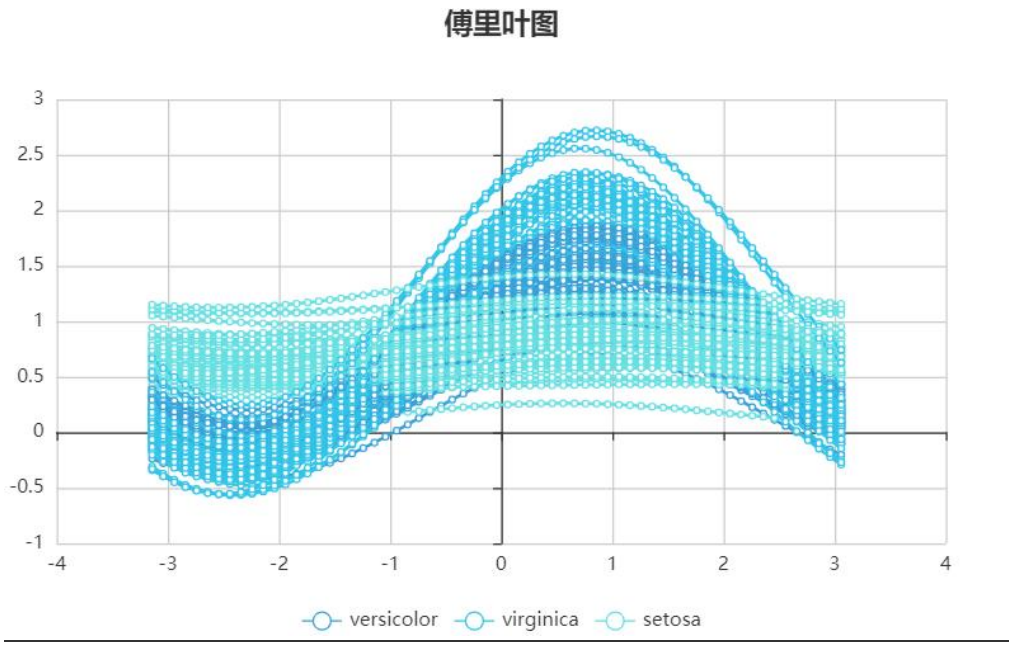
3.3.1 RadViz 雷达图高维数据显示



iris 数据集的 RadViz 雷达图-图 3.4.1

基本原理：对所有特征进行归一化之后，通过特征数量为每一个特征在 360 度均分为不同的轴作为不同特征（维度）的基底向量，将每一个的样本的所有基底向量叠加起来即得到了 RadViz。

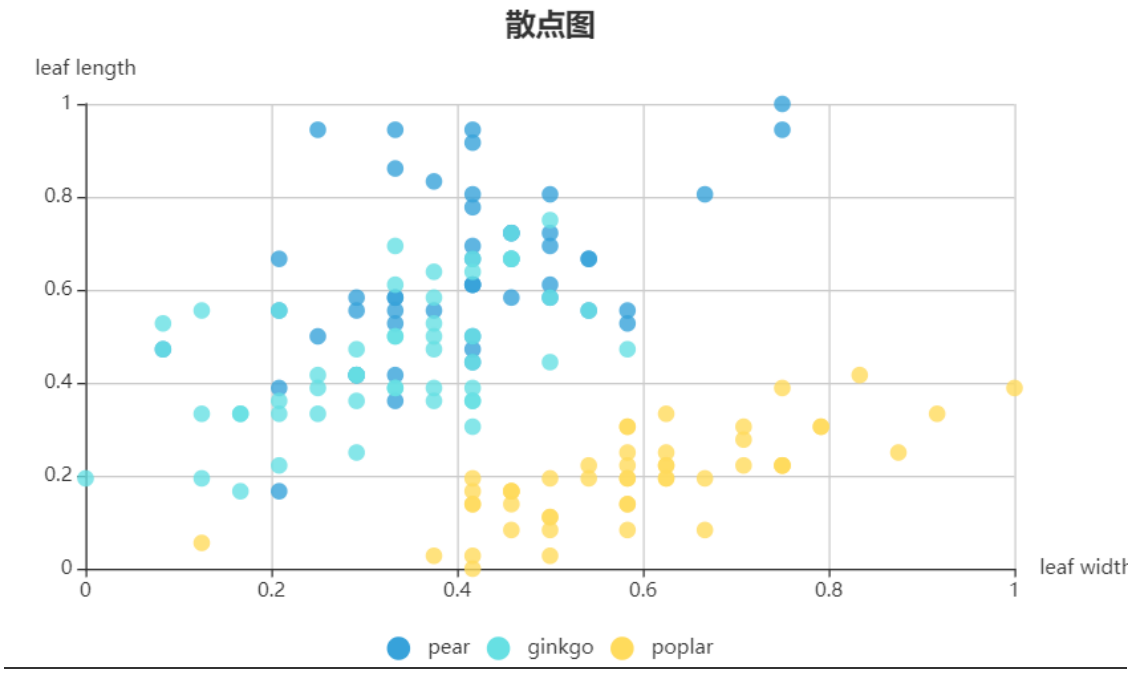
3.3.2 安德烈曲线傅里叶级数图高维数据显示



iris 数据集的安德烈曲线傅里叶级数图-图 3.4.2

生成一个有限项数的傅里叶级数，如何为每一个项的系数指派为一个特征，以此为每一个样本生成一个周期内的傅里叶级数。每一条曲线都代表一个样本，曲线与曲线之间的分离长度象征不同种类之间的差异程度。

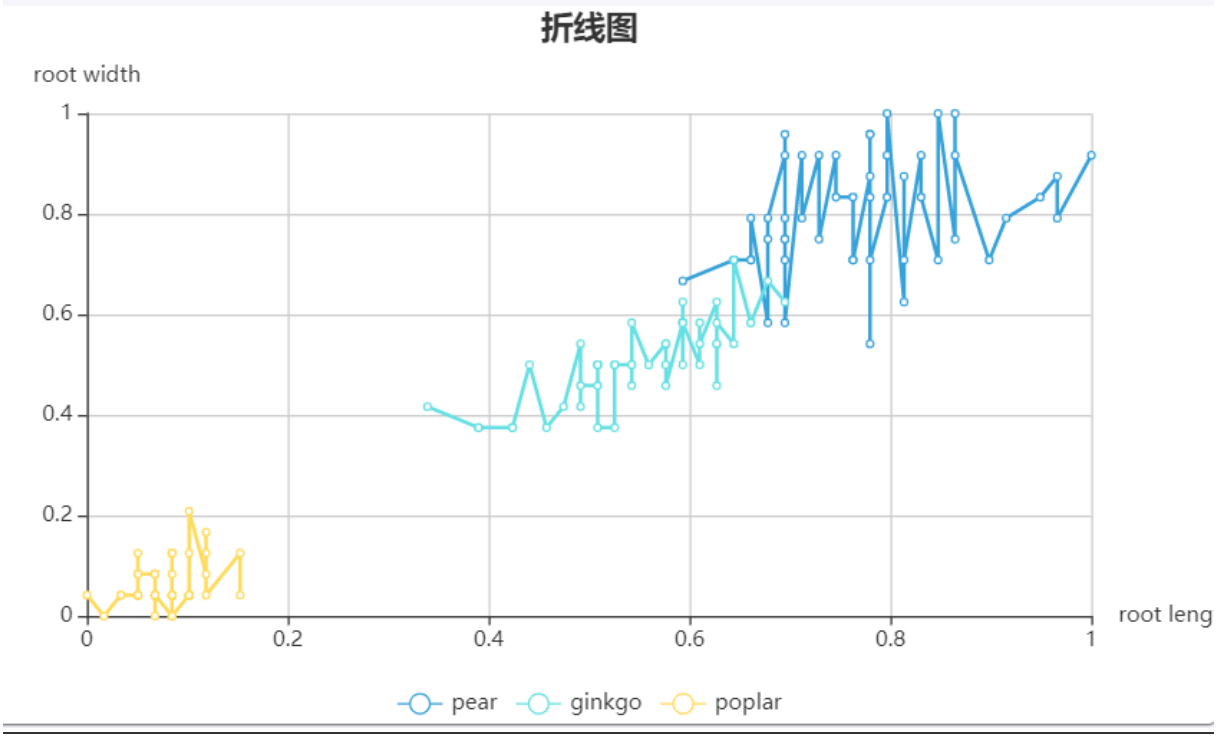
3.3.3 散点图高维数据显示



iris 数据集的散点图-图 3.4.3

每次随机选取数据集中的两个连续特征，作为 x 轴与 y 轴生成散点图，以显示种类分布受到所有不同特征的二维支配，通过低维映射高维的方式展现出联合分布。

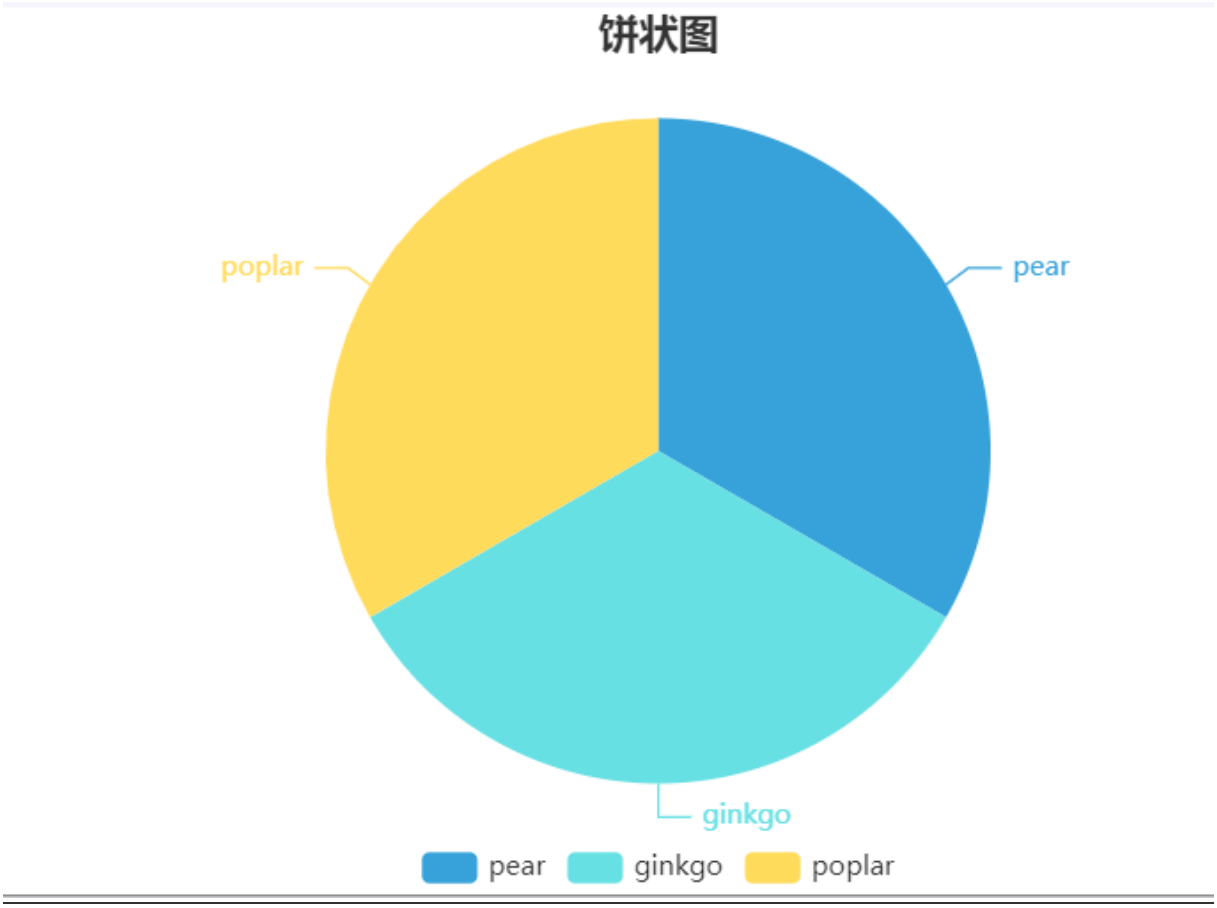
3.3.4 折线图高维数据显示



iris 数据集的折线图-图 3.4.5

每次随机选取数据集中的两个连续特征，作为 x 轴与 y 轴生成折线图，以显示种类分布受到所有不同特征的二维支配，通过低维映射高维的方式展现出联合分布。本质上是 3.4.3 散点图的升级版，除了希望得到种类与两个特征的联合分布以外，更要得出特征内潜在的变化（指上升、下降）关系。

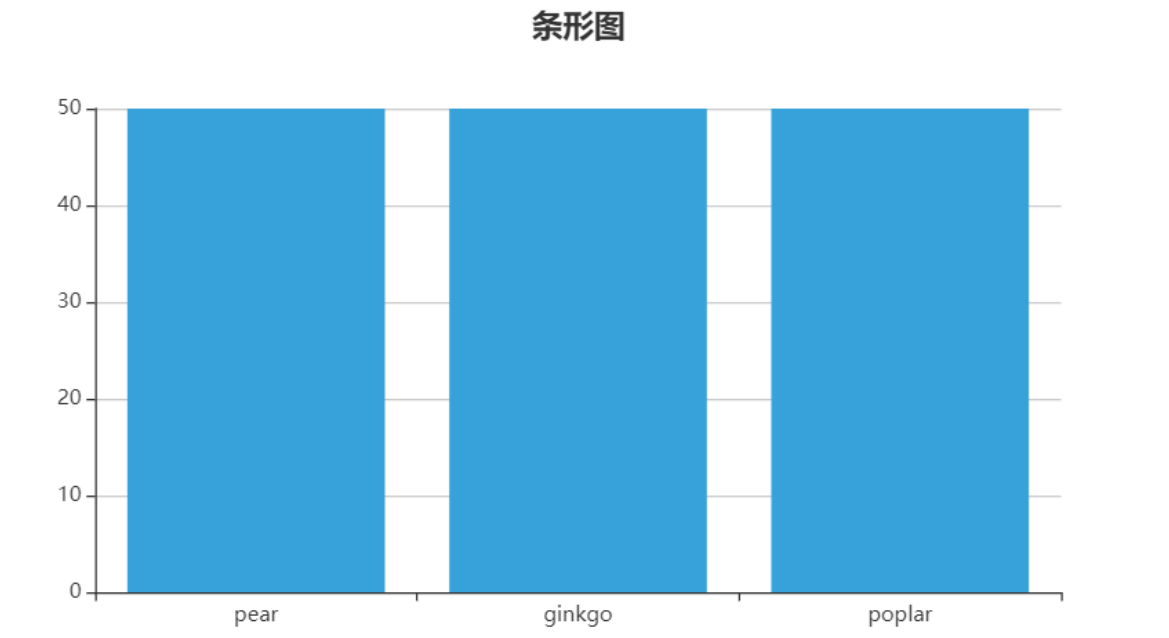
3.3.5 饼状图数据显示



iris 数据集的饼状图-图 3.4.5

显示出数据集所有的种类，以看出整个数据集的分类情况。

3.3.6 条形图数据显示

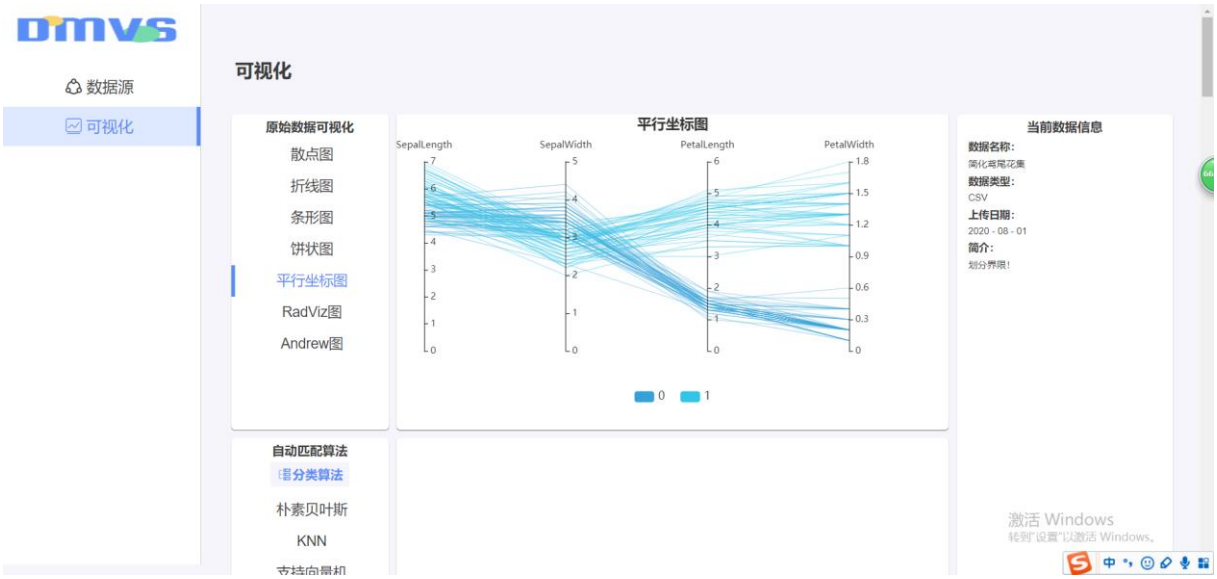


iris 数据集的条形图-图 3.4.6

显示出数据集所有的种类，以看出整个数据集的分类情况。

4. 系统测试

打开网站，出现的是 显示选择数据集的页面，有多种数据集供用户选择。左上角可以选择切换为可视化页面，右上角可以搜索数据源名称



在切换成可视化界面后，用户可以选择多种方式对刚才选择的数据进行可视化。

用户点击提交和预处理，和进行训练后，系统会智能选定算法对数据进行训练并建立模型。

稍等几许，便可以看到数据挖掘算法的可视化结果。

5. 结论

在这几天中，我们实现了数据挖掘可视化系统，并且实现了基本的功能，可以让用户直观的感受到可视化的原始数据以及经过数据挖掘后的数据。但是，还有一些功能，我们虽然有设想，但迫于时间原因没能够实现。由于第一次做项目，经验不足再加上线上信息沟通不便，导致团队协作十分低效。有了这一次的教训和经验，团队之间会更加磨合，更有信心面对下一次挑战。

6. 参考文献

[1] 王瑞松，林兰芬，大数据环境下时空多维数据的可视化研究，2016

[2] 任志伟，黄景涛，面对数据驱动建模的数据预处理方法研究，2013

[3] 李学学，彭珍瑞，基于数据预处理和回归分析技术的数据挖掘算法及其应用研究，2014

[4] 陈洁，宗平，数据挖掘分类算法的改进研究，2018