这里是标题

摘　要

摘要内容。

摘要内容。

在摘要里面插入数学 $\Delta = b^2 - 4ac$。

关键字： TeX　图片　表格　公式

一、 问题重述

1.1 问题背景

这里是一行正文。

1.2 问题要求

这里是一行引用的正文[1]。

二、 问题分析

2.1 问题一分析

详见图 1.

2.2 问题二分析

参见表 1.

前缀	端口	地址范围	地址数量
00	0	0000 0000 - 0011 1111	$2^6 = 64$
010	1	0100 0000 - 0101 1111	$2^5 = 32$
011	2	0110 0000 - 0111 1111	$2^6 + 2^5 = 96$
10		1000 0000 - 1011 1111	
11	3	1100 0000 - 1111 1111	$2^6 = 64$

表 1　一个表格示例

2.3 问题三分析

问题三分析。

2.4 问题四分析

问题四分析中文文献引用[2]。

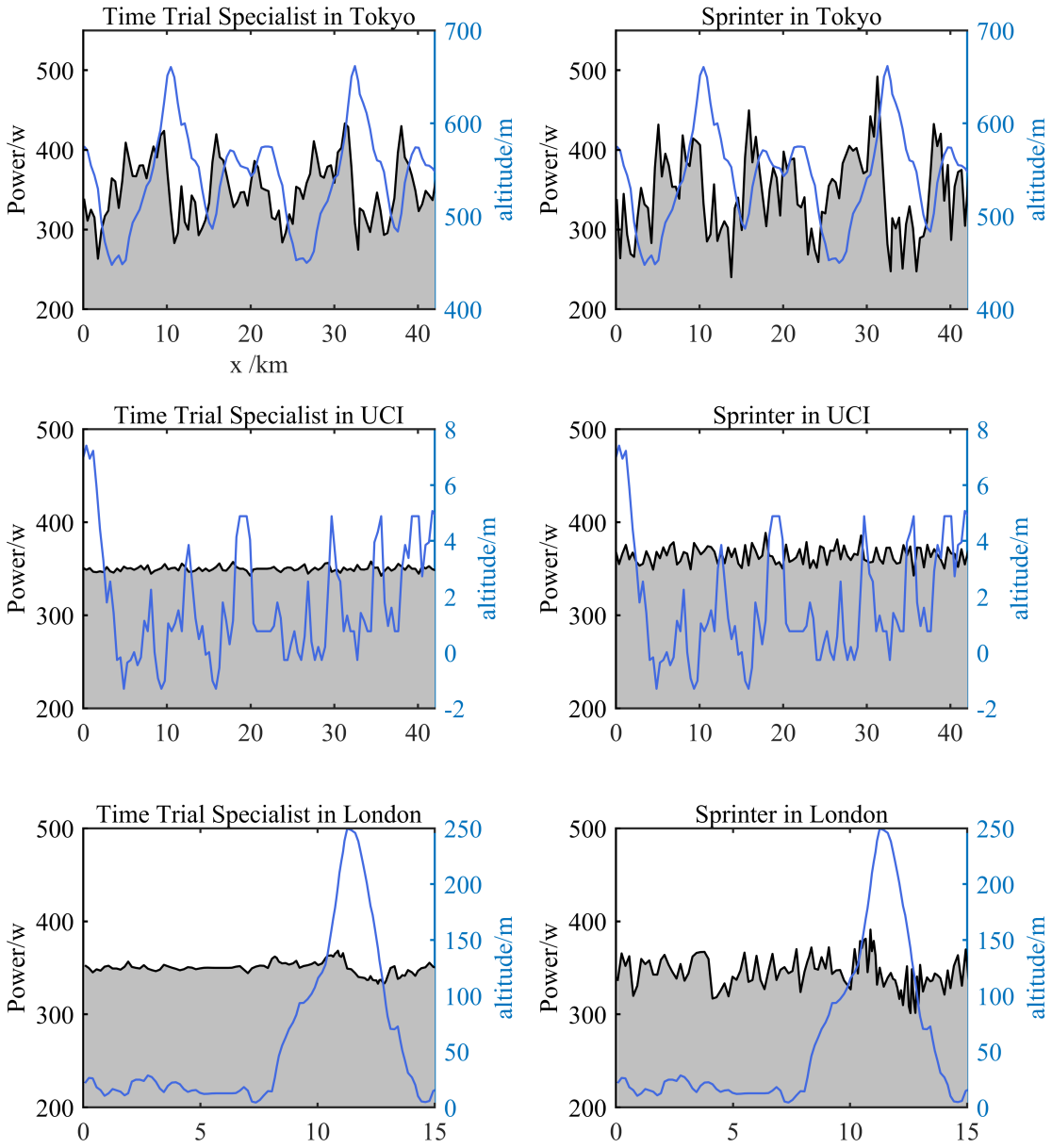

图 1　一个炒鸡大的插图示例

三、 模型假设

这里是模型假设

四、 符号说明

符号	说明	单位
Δ	0	-
p	功率	W

表 2　符号说明表

五、 数据预处理

我们进行了数据预处理。

六、 模型的建立与求解

6.1 问题一

6.1.1 模型的建立

align 数学环境

$$-ru_{i-1,j+1}+2(1+r)u_{i,j+1}-ru_{i+1,j+1}=u_{i-1,j}+2(1-r)u_{i,j}+ru_{i+1,j} \tag{1}$$

$$(1+\beta\triangle x)u_{1,j+1}-u_{2,j+1}=\beta\Delta xu_s(j+1) \tag{2}$$

$$(1+\beta\triangle x)u_{m,j+1}-u_{m-1,j+1}=\beta\Delta xu_s(j+1) \tag{3}$$

6.1.2 算法描述

算法伪代码环境

Algorithm 1 算法伪代码环境

Input: The original signal x.
Output: The energy-time-frequency distribution of x.

```
function EMD(x, seg_len)
    N ← length(x) / seg_len;
    for i=1 → i=N do
        seg(i) ← x(1+(i-1)*seg_len : i*seg_len);
    end for
end function
```

6.1.3 模型的求解

equation 等式环境测试

$$D = \sqrt{\frac{1}{N}\sum_{i=0}^{N}\left|u\left(2t'' - \left(t' + i\Delta t\right)\right) - u\left(t' + i\Delta t\right)\right|^2} \tag{4}$$

6.1.4 结果与分析

这里是求解结果 $q = p \times a$ 行内公式。

6.2 问题二

6.2.1 模型的建立

6.2.2 模型的求解

6.3 问题三

6.3.1 模型的建立

6.3.2 模型的求解

七、 模型评价与推广

7.1 模型评价

7.1.1 模型稳定性分析

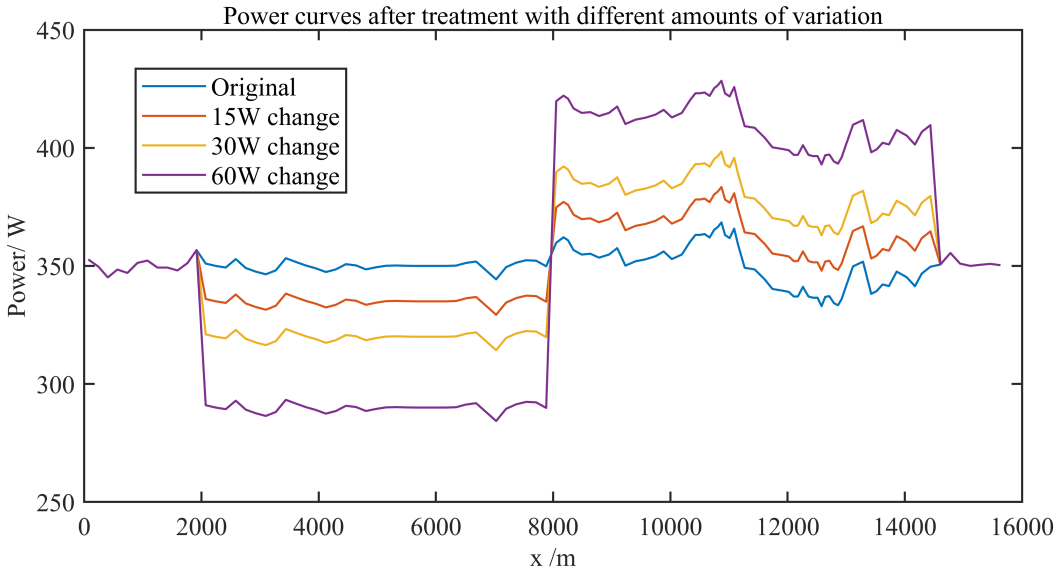

图 2　一般插图

7.1.2 模型的优点

- 无序列表测试
- a

7.2 模型的缺点

1. 有序列表测试
2. b

7.3 模型推广

一段正文。

两段正文。

参考文献

[1] MARTIN J S, HADFIELD S. Medusa: Universal Feature Learning via Attentional Multi-tasking[C] // CVPR 2022. .

[2] 王坤峰, 苟超, 段艳杰, 等. 生成式对抗网络 GAN 的研究进展与展望 [J]. 自动化学报, 2017, 43(3) : 321 – 332.

附录 A　代码文件列表

文件名	功能描述
Data.mat	附件数据

附录 B　代码

problem1.py 用来处理 · · · 逻辑。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from collections import Counter

train_frame = pd.read_csv('train_titanic.csv')

n = 10
# Bootstrap 采样
train_array = np.array(train_frame)
lenTrain, lenTrainFea = train_array.shape
m = int(lenTrain / 3)
rdmTrain = []
index = [i for i in range(0, lenTrain)]
index = np.array(index)

for i in range(0, n):
    rdmTmp = np.zeros((m, lenTrainFea))
    tmpindex = np.random.choice(index, size=m, replace=True)
    rdmTmp = train_array[tmpindex, :]
    # for j in range(0, m):
    #     tmpindex = np.random.choice(index, replace = True, size = 1)
    #     rdmTmp[j] = train_array[tmpindex, :]
    rdmTrain.append(rdmTmp)
# rdmTrain = np.array(rdmTrain)
# print(rdmTrain[n - 1])


def entropy(label):
    counter = Counter(label)
    ent = 0
    listCo = list(counter)
    length = len(label)
    for i in listCo:
        p = (counter[i] / length)
        ent -= p * math.log2(p)
    return ent
```

```
39
40
41  def split(feature, label, dimension):
42      # print(feature)
43      featureArr = np.array(feature)
44      # print(featureArr.shape)
45      # print(dimension)
46      dimension = int(dimension)
47      # dimArr =
48      counter = Counter(featureArr[:, dimension])
49      listCounter = list(counter)
50      count = len(feature)
51      split_feature = [[] for i in range(0, len(listCounter))]
52      split_label = [[] for i in range(0, len(listCounter))]
53      index = 0
54      for i in listCounter:
55          for j in range(0, count):
56              if feature[j][dimension] == i:
57                  split_feature[index].append(feature[j])
58                  split_label[index].append(label[j])
59          index += 1
60      return split_feature, split_label
61
62
63  def best_split(D, A):
64
65      # A‘维数
66      D = np.array(D)
67      frame = pd.DataFrame(A)
68      lenTrain = D.shape[0]
69      label = D[:, D.shape[1] - 1]
70      k = max(int(math.log2(len(A))), 1)
71      feature = np.zeros((lenTrain, 1))
72      # print(len(A))
73      # print(k)
74      tmp = frame.sample(n=k)
75      tmp = np.array(tmp)
76      # feature = label
77
78      for i in tmp:
79          feature = np.c_[feature, D[:, i]]
80      featureArr = feature[:, 1:k + 1]
81      sizeFeature = k
82      length = featureArr.shape[0]
83      best_entropy = 0
84      best_dimension = -1
85
86      ent = entropy(label)  # 总信息熵
87
88      for i in range(0, sizeFeature):
89          # 遍历所有分割
```

```
        splitFeature, splitLabel = split(feature, label, dimension=i)
        entNow = 0  # 当前总信息熵
        numSplite = len(splitFeature)
        for j in range(0, numSplite):
            entTemp = entropy(splitLabel[j])
            entNow += len(splitFeature[j]) / length * entTemp

        # 信息增益
        delta = ent - entNow
        if delta > best_entropy:
            best_entropy = delta
            best_dimension = i
        best_dimension = int(tmp[best_dimension])
    return best_dimension


# 记下所有属性可能的取值
D = np.array(train_frame)
A = set(range(D.shape[1] - 1))
possible_value = {}
for every in A:
    possible_value[every] = np.unique(D[:, every])

# 树结点类


class Node:
    def __init__(self, isLeaf=True, label=-1, index=-1):
        self.isLeaf = isLeaf  # isLeaf表示该结点是否是叶结点
        self.label = label  # label表示该叶结点的label（当结点为叶结点时有用）
        self.index = index  # index表示该分支结点的划分属性的序号（当结点为分支结点时有用）
        self.children = {}  # children表示该结点的所有孩子结点，dict类型，方便进行决策树的搜索

    def addNode(self, val: int, node):
        val = int(val)
        self.children[val] = node  # 为当前结点增加一个划分属性的值为val的孩子结点

# 决策树类


class DTree:
    def __init__(self):
        self.tree_root = None  # 决策树的根结点
        self.possible_value = {}  # 用于存储每个属性可能的取值

    '''
    TreeGenerate函数用于递归构建决策树，伪代码参照课件中的“Algorithm 1 决策树学习基本算法”
    '''
```

```
138
139     def TreeGenerate(self, D, A: set):
140
141         # 生成结点 node
142         node = Node()
143
144         feature = D[:, range(0, D.shape[1] - 1)]
145         label = D[:, D.shape[1] - 1]
146         counterlab = Counter(label)
147         listCo = list(counterlab)
148
149         # if D中样本全属于同一类别C then
150         #     将node标记为C类叶结点并返回
151         # end if
152         if len(listCo) == 1:
153             node.isLeaf = True
154             node.label = listCo[0]
155             return node
156
157         # if A = ∅ OR D中样本在A上取值相同 then
158         #     将node标记叶结点，其类别标记为D中样本数最多的类并返回
159         # end if
160         leng = 0
161         for i in A:
162             leng += len(set(feature[:, i])) - 1
163         if feature.shape[1] == 0 or leng == 0:
164             node.isLeaf = True
165             node.label = counterlab.most_common(1)[0][0]
166             return node
167
168         # 从A中选择最优划分属性a_star
169         # （选择信息增益最大的属性，用到上面实现的best_split函数）
170         a_star = best_split(D, A)
171
172         # for a_star 的每一个值a_star_v do
173         #     为node 生成每一个分支；令D_v表示D中在a_star上取值为a_star_v的样本子集
174         #     if D_v 为空 then
175         #         将分支结点标记为叶结点，其类别标记为D中样本最多的类
176         #     else
177         #         以TreeGenerate(D_v,A-{a_star}) 为分支结点
178         #     end if
179         # end for
180
181         node.isLeaf = False
182         node.index = a_star
183         # print(feature)
184         # print(a_star)
185         splitFea, splitLab = split(feature, label, a_star)
186         allType = self.possible_value[a_star]
187         lenSp = len(splitLab)
188         for i in allType:
```

```
            newNode = Node()
            node.addNode(i, newNode)
            node.children[i].isLeaf = True
            node.children[i].label = counterlab.most_common(1)[0][0]

        for i in range(0, lenSp):
            Dv = np.c_[splitFea[i], splitLab[i]]
            Av = set.copy(A)
            Av.remove(a_star)
            node.children[splitFea[i][0][a_star]] = self.TreeGenerate(Dv, Av)

        return node

    '''
    train函数可以做一些数据预处理（比如Dataframe到numpy矩阵的转换，提取属性集等），并调用TreeGenerate函数来递归地生成决策树
    '''

    def train(self, D):
        D = np.array(D)  # 将Dataframe对象转换为numpy矩阵（也可以不转，自行决定做法）
        A = set(range(D.shape[1] - 1))  # 属性集A

#         #记下每个属性可能的取值
        for every in A:
            self.possible_value[every] = np.unique(train_array[:, every])

        # 递归地生成决策树，并将决策树的根结点赋值给self.tree_root
        self.tree_root = self.TreeGenerate(D, A)

    '''
    predict函数对测试集D进行预测，输出预测标签
    '''

    def predict(self, D, i):
        D = np.array(D)   # 将Dataframe对象转换为numpy矩阵

        #lenTest = D.shape[0]
        #for i in range(0, lenTest):
        x = self.tree_root
        while x.isLeaf == False:
            # 向下查找
            x = x.children[D[i][x.index]]
        res = x.label

        return res
# ----- Your code here -------


dt = [DTree() for i in range(0, n)]
for i in range(0, n):
```

```
    dt[i].train(rdmTrain[i])
test_frame = pd.read_csv('test_titanic.csv')

# ----- Your code here -------
right = 0
testArr = np.array(test_frame)
lenTest, lenFea = testArr.shape
lenFea -= 1
for i in range(0, lenTest):
    true = 0
    predict = 0
    for j in range(0, n):
        if dt[j].predict(testArr, i) > 0:
            #print('1')
            true += 1
    if true > lenFea / 2:
        predict = 1
    print(predict, end='\t')
    if predict == testArr[i][lenFea]:
        right += 1
print(right / lenTest)
```

problem1.m 用于处理 ··· 逻辑。

```
function output = antiVpx(power,Mass,theta)
%ANTIVPX 此处显示有关此函数的摘要
%   此处显示详细说明
output = (-1).*Vpx(power, Mass, theta);
end
```