**计算机科学与技术学院**

数据结构大作业报告

**题 目： 天气预测系统**

**班 级：**

**姓 名：**

**学 号：**

# 一、需求分析与功能描述

**1.系统开发的背景和意义**

在现代社会，天气对人们日常生活对影响之处比比皆是，而天气状况是影响诸多事情比如出行决策的重要因素之一。比如不同的天气条件可能会对出行方式、行程安排和出行时间等方面产生重要影响。因此，开发一个基于天气状况帮助人们决策的系统具有重要的背景意义。本系统所预测的事项可以是多种多样的（甚至因素也可以不局限于天气，或者预测结果为不同天气），为方便读者理解，本系统将以天气预测出行为例展开阐述。

该系统的开发具有以下几个重要意义：

1. 提供个性化的出行建议：每个人对天气的感受和对出行方式的偏好各不相同。基于天气状况判断出行的系统可以根据用户的个人喜好和天气状况，提供个性化的出行建议。这样，用户可以更好地根据自己的需求和喜好做出决策，提高出行的满意度。

2. 提高出行效率和舒适度：天气状况对出行方式的选择有着重要影响。例如，在恶劣的天气条件下，选择乘坐公共交通工具或打车可能更加安全和舒适。而在良好的天气条件下，骑自行车或步行可能更加便捷和健康。基于天气状况判断出行的系统可以帮助用户根据实时天气信息做出明智的选择，提高出行效率和舒适度。

3. 促进可持续出行：气候变化和环境保护成为当今社会面临的重要问题。基于天气状况判断出行的系统可以鼓励人们选择更环保的出行方式。例如，在良好的天气条件下，推荐步行、骑自行车或使用公共交通工具，减少汽车使用，从而减少碳排放和交通拥堵，促进可持续出行。

4. 提供便捷的决策工具：基于天气状况判断出行的系统可以为用户提供便捷的决策工具。用户只需输入当前天气情况，系统就能根据事先设定的决策规则给出出行建议，省去了用户自行分析和判断的麻烦。这对于那些时间紧迫或对天气了解有限的用户尤为重要。

综上所述，基于天气状况判断出行的系统在提供个性化出行建议、提高出行效率和舒适度、促进可持续出行以及提供便捷决策工具等方面具有重要的背景和意义。这样的系统可以帮助人们做出更明智的出行决策，提高出行的质量和便利性。而系统本身具有较强的扩展性，可以适用于多种任务的预测。

1. 主要功能

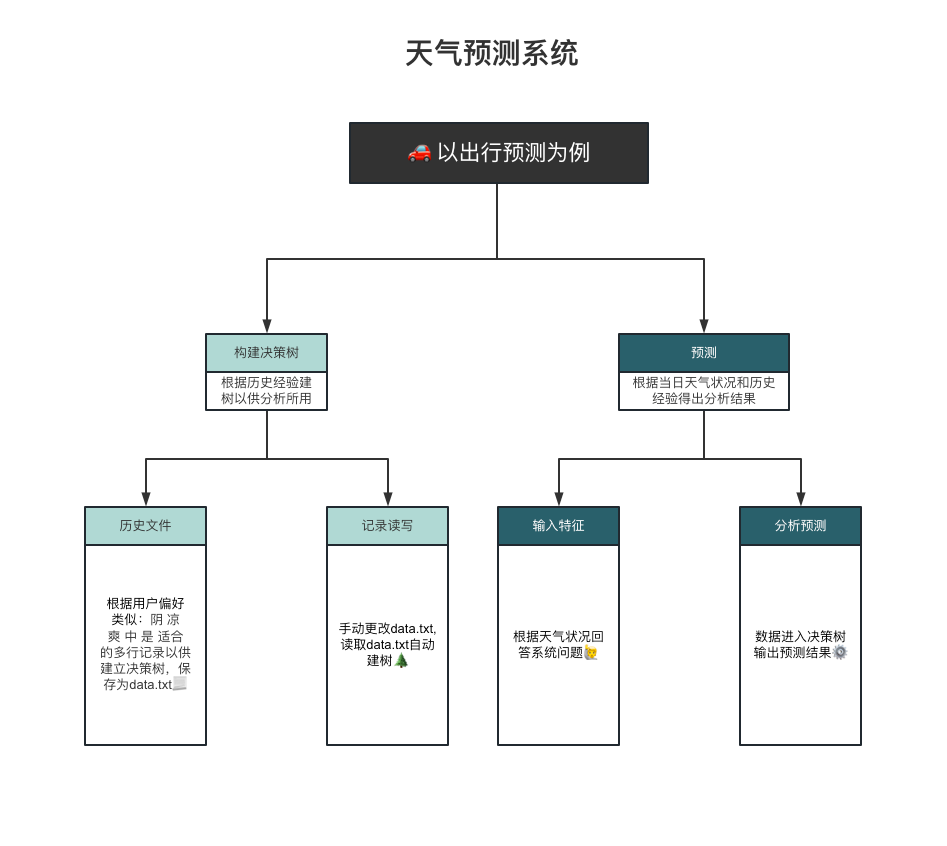


图1功能结构图

# 二、数据结构的选择与设计

## （一）系统中的数据描述

数据存储与data.txt中，为预测所需的经验信息，以天气预测是否适合出行为例，data.txt内容为：

晴 热 高 否 不适合

晴 热 高 是 不适合

阴 热 高 否 适合

雨 温 高 否 适合

雨 凉爽 中 否 适合

雨 凉爽 中 是 不适合

阴 凉爽 中 是 适合

晴 温 高 否 不适合

晴 凉爽 中 否 适合

雨 温 中 否 适合

晴 温 中 是 适合

阴 温 高 是 适合

阴 热 中 否 适合

雨 温 高 是 不适合

## 数据的存储结构

存储决策树的数据结构使用了链式存储结构。具体来说，它使用一个结构体 `**Node`** 表示决策树的节点，每个节点包含以下几个成员变量：

**`attribute`**：表示该节点的属性名称，即决策树的划分依据。

**`val`**：表示该节点的属性取值，即在该属性下的一个分支。

**`isLeaf`**：表示该节点是否是叶子节点，即是否是最终的分类结果。

**`childs`**：表示该节点的子节点列表，即根据不同属性取值的分支。

决策树的根节点由指针 **`root`** 指向。

**优点：**

1. 灵活性：链式结构可以根据决策树的实际情况动态地创建和释放节点，适应不同的决策树结构。

2. 可扩展性：链式结构便于在决策树的任意节点上添加新的分支或子树。

3. 易于理解和打印：链式结构直观地表示了决策树的层次关系，便于理解和可视化。

**缺点：**

1. 内存开销：链式结构需要为每个节点分配内存，可能会占用较多的内存空间，特别是当决策树较大时。

2. 存取效率：链式结构中访问节点需要通过指针的跳转，相对于线性存储结构，可能会稍微降低存取效率。

除了决策树，系统中的记录数据存储采用了线性结构：二维向量（vector）的存储结构来表示数据集。具体来说，数据集被表示为一个二维的向量，其中每一行表示一个数据样本，每一列表示一个特征或类别标签。

以下是该存储结构的优点和缺点：

**优点**：

1. 灵活性：向量是动态数组，可以根据需要动态调整大小，适应不同大小的数据集。

2. 简单直观：向量的索引操作和遍历操作非常直观，易于理解和使用。

3. 内存连续性：向量中的数据在内存中是连续存储的，这有助于提高数据的访问效率。

**缺点：**

1. 内存占用：向量在内存中占用连续的空间，如果数据集很大，可能会占用较多的内存空间。

2. 插入和删除操作效率较低：由于向量的内存连续性，插入和删除操作需要移动后续元素，效率较低。

3. 不适用于高维稀疏数据：如果数据集是高维稀疏数据，向量的存储方式可能会浪费大量的内存空间。

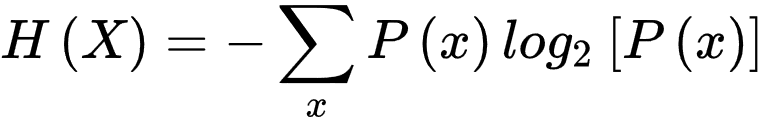
二维向量作为数据存储结构在简单的决策树分类器中是一种简单而有效的选择。它提供了灵活性和直观性，适用于中小规模的数据集。但对于大规模、高维稀疏数据集，可能需要考虑其他更高效的存储结构。

# 三、算法的选择与设计

为实现基于天气特征的预测，我使用一个简单的决策树分类器，用于根据输入的特征值预测是否适合进行某项活动。代码中使用了信息增益作为划分数据集的准则，并通过递归构建决策树来进行分类。以下是系统的算法简介：

1. 创建数据集：代码提供了一个示例数据集，也可以从文件中读取数据集。数据集由特征和对应的类别标签组成。

1. 计算香农熵：根据给定数据集，计算数据集的香农熵，用于衡量数据集的无序度。



香农熵是用来衡量数据集的无序度或者混乱程度的指标。在决策树分类器中，我们使用香农熵来选择最佳的特征进行划分。

香农熵的计算步骤如下：

* 统计数据集中每个类别的样本数量。
* 对于每个类别，计算其在整个数据集中的出现概率。
* 对于每个类别的概率，计算其对应的信息量
* 对所有类别的信息量求和，得到数据集的香农熵。

在代码中，我们首先统计数据集中每个类别的样本数量，并存储在字典 label\_counts 中。然后，利用这些统计信息，计算每个类别的概率，并根据概率计算对应的信息量。最后，将所有类别的信息量相加，得到数据集的香农熵。

3. 按照给定特征划分数据集：根据给定的特征和特征值，将数据集划分为子数据集。

4. 选择最好的特征进行划分：根据信息增益准则，选择最好的特征作为当前节点的划分特征。

1. 构建决策树：使用递归方法构建决策树，直到满足停止条件。每个节点包含一个特征和对应的取值，以及子节点列表。

* 输入参数为当前节点、数据集和特征列表。
* 如果当前数据集中的样本属于同一类别，则将当前节点标记为叶子节点，并将该类别作为节点的分类结果。
* 如果所有特征都已经遍历完，但数据集中仍然存在多个类别，则将当前节点标记为叶子节点，并采用多数表决的方法确定分类结果。
* 否则，选择最佳特征作为当前节点的划分特征。
* 遍历最佳特征的每个取值，创建一个子节点，并将数据集划分为对应取值的子集。
* 递归调用构建决策树函数，将子节点作为当前节点的子节点，并更新特征列表（去除已使用的特征）。
* 返回当前节点。

6. 打印决策树：将构建好的决策树以可读的形式打印出来。

7. 预测分类：根据输入的特征值，利用构建好的决策树进行分类预测。

1. 释放节点：在使用完决策树后，释放节点的内存。

* createDataset 函数用于创建数据集和特征列表。数据集 X 是一个二维向量，存储了样本数据，特征列表 attributes 存储了特征的名称。
* calcShanno 函数用于计算给定数据集的香农熵。它统计了数据集中每个类别的样本数量，并根据概率计算香农熵。
* splitDataSet 函数根据给定特征和特征值划分数据集，返回划分后的子数据集。
* createFeatureList 函数根据给定特征在数据集中创建特征列表，即该特征的所有可能取值。
* chooseBestFeatureToSplit 函数选择最好的特征进行数据集划分。它计算每个特征的信息增益，并选择信息增益最大的特征作为划分依据。
* majorityCnt 函数返回出现次数最多的类别名称，用于处理当数据集已处理所有属性但类别标签不唯一的情况。
* Node 结构体表示决策树的节点，包含属性名称、属性取值、是否是叶子节点以及子节点列表。
* createTree 函数递归地构建决策树。它根据数据集和特征列表选择最佳特征进行划分，并创建子节点。
* print 函数用于打印决策树的结构。
* classify 函数根据决策树和给定的测试数据进行分类预测。
* main 函数是程序的入口。它调用其他函数来创建决策树、打印决策树结构，并根据用户输入的测试数据进行分类预测。

这样的系统可以便于修改成其他任务的主要原因是它的模块化设计。可以通过修改数据集、特征和类别标签，以及修改选择最佳特征和分类预测的方法，来适应不同的任务。例如，可以根据不同的特征和类别标签来构建一个新的数据集，然后使用相同的决策树构建和分类方法来解决其他分类问题。这种灵活性使得代码可以方便地扩展和应用于不同的任务和数据集。

开始

读取data.txt

输入特征

是否合法

N

Y

得出分析结果

结束

图2 系统操作流程图

# 主要代码

//递归构建决策树

Node\* createTree(Node \*root, const vector< vector<string> > &data, vector<string> &attribute) {

if(root == NULL)

root = new Node();

vector<string> classList;

set<string> classList1;

int i, j;

int label = data[0].size() - 1;

int n = data.size();

for(i=0; i<n; i++) {

classList.push\_back(data[i][label]);

classList1.insert(data[i][label]);

}

//如果所有实例都属于同一类，停止划分

if(classList1.size() == 1) {

if(classList[0] == "适合")

root->attribute = "适合";

else

root->attribute = "不适合";

root->isLeaf = true;

return root;

}

//遍历完所有特征，返回出现次数最多的类别

if(data[0].size() == 1) {

root->attribute = majorityCnt(classList);

return root;

}

int bestFeatureIndex = chooseBestFeatureToSplit(data);

//得到属性的所有可能值

vector<string> featureList = createFeatureList(data, bestFeatureIndex);

string bestFeature = attribute[bestFeatureIndex];

//记录要划分的属性

root->attribute = bestFeature;

//对于当前属性的每个可能值，创建新的分支

for(i=0; i<featureList.size(); i++) {

vector<string> subAttribute;

for(j=0; j<attribute.size(); j++) {

if(bestFeature != attribute[j])

subAttribute.push\_back(attribute[j]);

}

Node \*newNode = new Node();

newNode->val = featureList[i];//记录属性的取值

createTree(newNode, splitDataSet(data, bestFeatureIndex, featureList[i]), subAttribute);

root->childs.push\_back(newNode);

}

return root;

}

# 五、测试结果

表1 系统测试用例及问题解决方法

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **测试输入** | **预期输出** | **实际输出** | **运行bug** | **解决方法** |
| **晴 热 高 否** | **不适合** | **不适合** | **没问题** |  |
| **晴 热 高 是** | **不适合** | **不适合** | **没问题** |  |
| **阴 热 高 否** | **适合** | **适合** | **没问题** |  |
| **雨 温 高 是** | **不适合** | **不适合** | **没问题** |  |

运行界面测试图如图3所示。

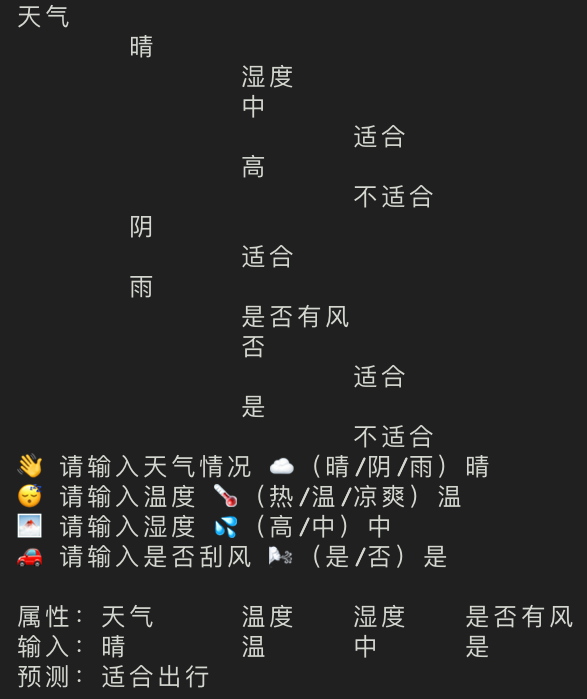


图3 系统运行界面

# 六、分析与总结

这个系统实现了一个基于决策树的天气情况决定出行决策功能。系统通过读取数据文件（data.txt）中的学生出行数据，使用决策树算法构建决策树模型，并根据用户输入的天气、温度、湿度和是否有风等特征值，预测学生是否适合出行。

在系统中使用了以下数据结构和算法：

数据结构：

- 向量（vector）：用于存储学生出行数据集、特征列表和类别列表等。

- 映射（map）：用于计算香农熵和统计类别出现次数。

- 集合（set）：用于创建特征的取值集合，以及统计类别的唯一值。

算法：

- 香农熵计算：用于衡量数据集的无序度和不确定性。

- 数据集划分：根据给定特征和特征值，将数据集划分为更小的子集。

- 选择最佳特征：基于信息增益，选择最佳特征来构建决策树。

- 递归构建决策树：根据数据集和特征列表，递归地构建决策树模型。

- 预测分类：根据构建的决策树模型，对新的输入进行分类预测。

在整个系统项目开发过程中，我学会了如何使用C++语言实现决策树算法，并将其应用于学生出行决策的场景中。我了解了决策树的原理和构建过程，以及如何使用信息增益选择最佳特征进行划分。此外，我还学会了如何读取和处理文件中的数据，并将其转换为合适的数据结构进行存储和处理。

在未来的专业学习中，我的目标是进一步深入学习和掌握数据科学和数据结构领域的知识和技能。我希望能够学习更多的机器学习算法、深度学习模型以及数据处理和特征工程的技术。我还计划学习相关的编程语言和工具，以便能够更好地实现和应用这些算法和模型。我希望能够在未来的职业生涯中，通过机器学习和数据挖掘技术，解决实际问题并取得更好的成果。