因为使用的是英文字母，所以所有的参数都应当从中文换成英文。

替换完成之后，考虑一下把决策树组件放在哪里。

不考虑了，直接放到CPP\_BaseCharacter里吧

现在需要回顾一下，决策树的相关函数与数据结构

struct DataRow

struct TreeNode

double calculateEntropy(const vector<DataRow>& dataset, int targetIdx)

double calculateInfoGain(const vector<DataRow>& dataset, int attrIdx, int targetIdx)

int chooseBestAttribute(const vector<DataRow>& dataset, const vector<bool>& usedAttributes, int targetIdx)

shared\_ptr<TreeNode> buildDecisionTree(const vector<DataRow>& dataset,const vector<string>& attributes,vector<bool> usedAttributes,int targetIdx)

string predict(const shared\_ptr<TreeNode>& root, const DataRow& sample, const vector<string>& attributes)

进入到main函数中，

vector<DataRow> dataset =

vector<string> attributes =

vector<shared\_ptr<TreeNode>> trees

DataRow sample =

输出结果是

第一个预测：predict(trees[0], sample, attributes)

第二个预测：predict(trees[1], sample, attributes)

第三个预测：predict(trees[2], sample, attributes)

所以我们需要在U++里实现这些功能

在C++中，部分自带的数据结构（如vector、string等）并不能直接在U++中使用，我们必须使用改封装的U++数据结构

另外，结构的定义应当遵循UnrealEngine的编码标准规则，我们需要让结构的名称第一个字母是F

TArray--vector

FString--string

TMap--map

USTRUCT()

struct FDataRow

{

    GENERATED\_BODY()

    TArray<FString> Conditions;

    TArray<FString> Decisions;

};

USTRUCT()

struct FTreeNode

{

    GENERATED\_BODY()

    FString Attribute;

    TMap<FString, TSharedPtr<FTreeNode>> Children;

    FString Decision;

};

double ACPP\_BaseCharacter::CalculateEntropy(const TArray<FDataRow>& Dataset, int32 TargetIndex)

{

    TMap<FString, int32> Freq;

    for (const auto& Row : Dataset)

    {

        Freq.FindOrAdd(Row.Decisions[TargetIndex])++;

// 下面一行是C++风格，上面一行是U++风格

        // Freq[Row.Decisions[TargetIndex]]++;

    }

    double Entropy = 0.0;

    for (const auto& Pair : Freq)

    {

        double Prob = static\_cast<double>(Pair.Value) / Dataset.Num();

        Entropy -= Prob \* FMath::Log2(Prob);

    }

    return Entropy;

}

double ACPP\_BaseCharacter::CalculateInfoGain(const TArray<FDataRow>& Dataset, int32 AttrIdx, int32 TargetIdx)

{

    TMap<FString, TArray<FDataRow>> Subsets;

    for (const auto& Row : Dataset)

    {

        Subsets.FindOrAdd(Row.Conditions[AttrIdx]).Add(Row);

    }

    double SubsetEntropy = 0.0;

    for (const auto& Subset : Subsets)

    {

        double Weight = static\_cast<double>(Subset.Value.Num()) / Dataset.Num();

        SubsetEntropy += Weight \* CalculateEntropy(Subset.Value, TargetIdx);

    }

    return CalculateEntropy(Dataset, TargetIdx) - SubsetEntropy;

}

int32 ACPP\_BaseCharacter::ChooseBestAttribute(const TArray<FDataRow>& Dataset, const TArray<bool>& UsedAttributes,

    int32 TargetIdx)

{

    double MaxGain = -1;

    int32 BestAttr = -1;

    for (int32 i = 0; i < UsedAttributes.Num(); ++i)

    {

        if (!UsedAttributes[i])

        {

            double Gain = CalculateInfoGain(Dataset, i, TargetIdx);

            if (Gain > MaxGain)

            {

                MaxGain = Gain;

                BestAttr = i;

            }

        }

    }

    return BestAttr;

}

FString ACPP\_BaseCharacter::Predict(const TSharedPtr<FTreeNode>& Root, const FDataRow& Sample,const TArray<FString>& Attributes)

{

    if (!Root.IsValid())

    {

        UE\_LOG(LogTemp, Error, TEXT("Invalid decision tree root node!"));

        return TEXT("Invalid");

    }

    if (!Root->Decision.IsEmpty())

    {

        return Root->Decision;

    }

    int32 AttrIndex = Attributes.Find(Root->Attribute);

    if (AttrIndex == INDEX\_NONE)

    {

        UE\_LOG(LogTemp, Warning, TEXT("Attribute %s not found in attributes list"), \*Root->Attribute);

        return TEXT("Unknown");

    }

    if (!Sample.Conditions.IsValidIndex(AttrIndex))

    {

        UE\_LOG(LogTemp, Warning, TEXT("Sample data missing attribute at index %d"), AttrIndex);

        return TEXT("Unknown");

    }

    const FString& AttrValue = Sample.Conditions[AttrIndex];

    const TSharedPtr<FTreeNode>\* ChildNodePtr = Root->Children.Find(AttrValue);

    if (!ChildNodePtr || !ChildNodePtr->IsValid())

    {

        UE\_LOG(LogTemp, Warning, TEXT("No child node found for attribute value: %s"), \*AttrValue);

        return TEXT("Unknown");

    }

    return Predict(\*ChildNodePtr, Sample, Attributes);

}

这里我们在实现BuildDecisionTree（）函数的时候，需要创建一个函数，实现map.begin()的功能，因为TMap没有设个功能。

FString ACPP\_BaseCharacter::GetPluralityClass(const TMap<FString, int32>& ClassCounts)

{

    if (ClassCounts.Num() == 0)

    {

        UE\_LOG(LogTemp, Warning, TEXT("Empty class counts in plurality calculation"));

        return FString();

    }

    FString MajorityClass;

    int32 MaxCount = 0;

    // 遍历统计结果

    for (const auto& Pair : ClassCounts)

    {

        if (Pair.Value > MaxCount)

        {

            MaxCount = Pair.Value;

            MajorityClass = Pair.Key;

        }

        else if (Pair.Value == MaxCount)

        {

            // 处理平票情况（可选：根据业务逻辑调整）

            MajorityClass = FString::Printf(TEXT("%s/%s"), \*MajorityClass, \*Pair.Key);

        }

    }

    return MajorityClass;

}

最后实现BuildDecisionTree函数

TSharedPtr<FTreeNode> ACPP\_BaseCharacter::BuildDecisionTree(const TArray<FDataRow>& Dataset,

    const TArray<FString>& Attributes, TArray<bool> UsedAttributes, int32 TargetIdx)

{

    // 参数有效性验证

    if (Dataset.IsEmpty() || Attributes.IsEmpty() || TargetIdx < 0 || TargetIdx >= Attributes.Num())

    {

        UE\_LOG(LogTemp, Error, TEXT("Invalid input parameters for decision tree construction"));

        return nullptr;

    }

    // 创建节点（使用虚幻的智能指针系统）

    TSharedPtr<FTreeNode> Node = MakeShared<FTreeNode>();

    // 类别计数（使用虚幻的容器）

    TMap<FString, int32> ClassCounts;

    for (const FDataRow& Row : Dataset)

    {

        // 数据有效性检查

        if (Row.Decisions.IsValidIndex(TargetIdx))

        {

            const FString& Decision = Row.Decisions[TargetIdx];

            ClassCounts.FindOrAdd(Decision)++;

        }

        else

        {

            UE\_LOG(LogTemp, Warning, TEXT("Invalid target index in data row"));

        }

    }

    // 终止条件1：所有样本属于同一类别

    if (ClassCounts.Num() == 1)

    {

        Node->Decision = ClassCounts.CreateIterator().Key();

        return Node;

    }

    // 选择最佳划分属性（虚幻风格参数传递）

    const int32 BestAttr = ChooseBestAttribute(Dataset, UsedAttributes, TargetIdx);

    // 终止条件2：没有可用属性或选择失败

    if (BestAttr == INDEX\_NONE || !Attributes.IsValidIndex(BestAttr))

    {

        Node->Decision = GetPluralityClass(ClassCounts); // 需要实现多数表决辅助函数

        return Node;

    }

    // 设置节点属性

    Node->Attribute = Attributes[BestAttr];

    // 更新已用属性（使用Unreal的内存布局优化）

    TArray<bool> NewUsedAttributes = UsedAttributes;

    if (NewUsedAttributes.IsValidIndex(BestAttr))

    {

        NewUsedAttributes[BestAttr] = true;

    }

    // 创建数据子集（使用虚幻的容器和算法）

    TMap<FString, TArray<FDataRow>> Subsets;

    for (const FDataRow& Row : Dataset)

    {

        if (Row.Conditions.IsValidIndex(BestAttr))

        {

            const FString& Key = Row.Conditions[BestAttr];

            Subsets.FindOrAdd(Key).Add(Row);

        }

    }

    // 递归构建子树

    for (auto& SubsetPair : Subsets)

    {

        const FString& AttributeValue = SubsetPair.Key;

        TArray<FDataRow>& SubsetData = SubsetPair.Value;

        if (SubsetData.IsEmpty())

        {

            // 创建叶节点

            TSharedPtr<FTreeNode> LeafNode = MakeShared<FTreeNode>();

            LeafNode->Decision = GetPluralityClass(ClassCounts);

            Node->Children.Add(AttributeValue, LeafNode);

        }

        else

        {

            // 递归调用（使用移动语义优化大数据集）

            TSharedPtr<FTreeNode> ChildNode = BuildDecisionTree(

                SubsetData,

                Attributes,

                NewUsedAttributes,

                TargetIdx

            );

            if (ChildNode.IsValid())

            {

                Node->Children.Add(AttributeValue, ChildNode);

            }

            else

            {

                UE\_LOG(LogTemp, Warning, TEXT("Failed to build child node for attribute value: %s"), \*AttributeValue);

            }

        }

    }

    return Node;

}

函数实现完成。

然后我们就需要开始调用这些函数

在C++中，在main中创建了如下几个变量

vector<DataRow> dataset //训练集

vector<string> attributes //参数

DataRow sample //测试示例

vector<shared\_ptr<TreeNode>> trees //三个决策树

for (int target = 0; target < 3; ++target)

{

        vector<bool> usedAttributes(5, false);

        trees.push\_back(buildDecisionTree(dataset, attributes, usedAttributes, target));

}

得到预测结果是

predict(trees[0], sample, attributes)

predict(trees[1], sample, attributes)

predict(trees[2], sample, attributes)

我们要是想将其转成U++，就要想将他们放在哪里。尤其是for循环，不能像放在main函数里一样直接放在类声明中。

所以我lizzy阿志是这样搞得：

在类声明中创建几个变量

TArray<FDataRow> dataset =

TArray<FString> attributes =

FDataRow sample =

TArray<TSharedPtr<FTreeNode>> Trees

然后创建一个函数

void SetTrees(TArray<TSharedPtr<FTreeNode>>& Root,TArray<FDataRow> Dataset,TArray<FString> Attributes);

这个函数负责for循环的东西：就是给Trees数组进行赋值

void ACPP\_BaseCharacter::SetTrees(TArray<TSharedPtr<FTreeNode>>& Root, TArray<FDataRow> Dataset,TArray<FString> Attributes)

{

    for(int32 TargetIndex = 0; TargetIndex < 3; ++TargetIndex)

    {

        TArray<bool> UsedAttributes;

        UsedAttributes.Init(false,5);

        TSharedPtr<FTreeNode> NewTree = BuildDecisionTree(Dataset,Attributes,UsedAttributes,TargetIndex);

        // 空指针安全检查

        if (NewTree.IsValid())

        {

            Root.Add(NewTree);

        }

    }

}

当我们需要得到预测结果时，原来的代码是

predict(trees[0], sample, attributes)

在U++中几乎没变

Predict(Trees[0],sample,attributes);

但是需要注意，此时我们要用到Trees的实际数值。

也就是说，在得到预测结果之前，我们就应当给Trees进行初始化，也就是进行一次SetTrees（）的调用

那就很简单了，在构造函数里调用一下就行

ACPP\_BaseCharacter::ACPP\_BaseCharacter()

{

    PrimaryActorTick.bCanEverTick = false;

    SetTrees(Trees,dataset,attributes);

}

这样就完成了决策树封装到U++中

然后要考虑的就是，在什么时候调用Predict函数来得到预测结果