Classification by Splitting Data: Decision Trees & Random Forest

CSCI316: Big Data Mining Techniques and Implementation

Table of Contents

---

1. Introduction to Classification

The Classification Problem

- **Definition**: Given a set of training records with known class labels, build a model to predict the class of new, unlabeled records
- **Process Flow**:
  - **Training Phase**: Learn model from training data (Induction)
  - **Testing Phase**: Apply model to test data (Deduction)

Example Dataset Structure

```
Tid | Attrib1 | Attrib2 | Attrib3 | Class
----|---------|---------|---------|-------
1   | Yes     | Large   | 125K    | No
2   | No      | Medium  | 100K    | No
3   | No      | Small   | 70K     | No
...
```

---

2. Decision Trees Fundamentals

What is a Decision Tree?

- **Structure**: A flowchart-like tree structure where:
  - **Internal nodes** (non-leaf nodes): Tests on attributes
  - **Branches**: Outcomes of tests
  - **Leaf nodes**: Class labels

Key Characteristics

- **Human-like Decision Making**: Simulates human decision-making process
- **Interpretability**: Easy to understand and explain
- **Flexibility**: Can handle both categorical and continuous attributes

Decision Tree Components

1. **Root Node**: Starting point of the tree
2. **Splitting Attributes**: Features used to partition data
3. **Decision Nodes**: Internal nodes that test attribute values
4. **Leaf Nodes**: Terminal nodes containing class predictions

Example Tree Structure

```
Refund
├── Yes → No (Class)
└── No → MaritalStatus
    ├── Single, Divorced → TaxableIncome
    │   ├── < 80K → No
    │   └── ≥ 80K → Yes
    └── Married → No
```

---

3. Tree Construction Algorithm

Hunt's Algorithm (Basic Framework)

Input: Dataset D, Attribute_List **Output**: Decision Tree

General Procedure for Node t with dataset Dt:

1. **Homogeneous Case**: If all records in Dt belong to same class yt
   - Make t a leaf node labeled as yt

2. **Empty Dataset**: If Dt is empty
   - Make t a leaf node with same class as parent
3. **No More Attributes**: If no attributes left to split
   - Make t a leaf node with majority class
4. **Continue Splitting**: Otherwise
   - Split dataset into smaller subsets
   - Recursively apply procedure to child nodes

Detailed Algorithm Steps

Tree Induction Algorithm

python

```python
def generate_decision_tree(D, Attribute_List):
    # Step 1: Create node N
    create_node(N)

    # Step 2-3: Check if all tuples have same class
    if all_same_class(D):
        return N as leaf with class C

    # Step 4-5: Check if attribute list is empty
    if Attribute_List.is_empty():
        return N as leaf with majority class

    # Step 6: Find best splitting attribute
    best_attr = find_best_split(D, Attribute_List)

    # Step 7: Update attribute list
    New_Attribute_List = Attribute_List - {best_attr}

    # Step 8-12: Create child nodes
    for each value v in best_attr:
        Dv = subset_with_value(D, best_attr, v)
        if Dv.is_empty():
            attach_leaf(N, majority_class(D))
        else:
            child = generate_decision_tree(Dv, New_Attribute_List)
            attach_child(N, child)

    return N
```

---

4. Splitting Criteria

Attribute Types and Splitting Methods

1. Nominal/Categorical Attributes

- **Multi-way Split**: Use as many partitions as distinct values
- **Binary Split**: Divide values into two subsets
  - Example: CarType = {Family, Sports, Luxury}
  - Multi-way: Family | Sports | Luxury
  - Binary: {Family, Sports} | {Luxury}

2. Ordinal Attributes

- **Discretization**: Convert to categorical
  - Static: Discretize once at beginning
  - Dynamic: Use percentiles, clustering, etc.
- **Binary Decision**: (A < v) or (A ≥ v)

3. Continuous Attributes

- **Binary Split**: Find optimal threshold
  - Example: TaxableIncome > 80K?
- **Multi-way Split**: Create multiple ranges
  - Example: <10K, [10K,25K), [25K,50K), [50K,80K), >80K

---

5. Impurity Measures

Why Measure Impurity?

- **Goal**: Prefer nodes with homogeneous class distributions
- **Principle**: Pure nodes (one class) are better than mixed nodes

1. Shannon Entropy (Information Gain)

Formula

```
H(D) = -Σ p(x) * log₂(p(x))
```

where p(x) is the probability of class x in dataset D

Conditional Entropy

```
H(D|P) = Σ (|Di|/|D|) * H(Di)
```

Information Gain

```
InfoGain(P) = H(D) - H(D|P)
```

Python Implementation

python

```python
def calcShannonEnt(dataSet):
    numEntries = len(dataSet)
    labelCounts = {}

    for featVec in dataSet:
        currentLabel = featVec[-1]
        if currentLabel not in labelCounts.keys():
            labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1

    shannonEnt = 0.0
    for key in labelCounts:
        prob = float(labelCounts[key]) / numEntries
        shannonEnt -= prob * log(prob, 2)

    return shannonEnt
```

2. Gini Index

Formula

```
Gini(D) = 1 - Σ pi²
```

where pi is the probability of class i

Gini for Split

```
GiniP(D) = Σ (|Di|/|D|) * Gini(Di)
ΔGini = Gini(D) - GiniP(D)
```

Characteristics

- Measures probability of misclassification
- Range: [0, 0.5] for binary classification
- 0 = pure node, 0.5 = maximum impurity

3. Gain Ratio

Purpose

- Addresses Information Gain's bias toward multi-valued attributes

Formula

```
SplitInfo(P) = -Σ (|Di|/|D|) * log(|Di|/|D|)
GainRatio = InfoGain(P) / SplitInfo(P)
```

4. Variance (Binary Classification)

Formula

```
Var(D) = p(1-p)
```

where p is probability of class 0

Characteristics

- Simple error measure for binary classification
- Maximum when p = 0.5
- Minimum when p = 0 or p = 1

Comparison of Impurity Measures

Measure	Advantages	Disadvantages
Information Gain	Simple, widely used	Biased toward multi-valued attributes
Gain Ratio	Reduces multi-valued bias	Prefers unbalanced splits
Gini Index	Good for equal-sized partitions	Difficulty with many classes
Variance	Suitable for binary classification	Limited to binary problems

---

6. Implementation Details

Python Data Structure

python

```python
# Decision tree as nested dictionary
tree = {
    index_of_splitting_feature: {
        value_0: subtree_0 or leaf_0,
        value_1: subtree_1 or leaf_1,
        ...
    }
}
```

Leaf Node Representation

python

```python
# Simple: just class label
leaf = "ClassA"

# Advanced: class frequency vector
leaf = np.array([q1, q2, ..., qm])  # qi = |DCi|
```

Best Split Selection

python

```python
def chooseBestMultiSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1
    baseEntropy = calcShannonEnt(dataSet)
    bestInfoGain = 0.0
    bestFeature = -1

    for i in range(numFeatures):
        uniqueVals = set([tuple[i] for tuple in dataSet])
        newEntropy = 0.0

        for value in uniqueVals:
            subDataSet = splitDataSet(dataSet, i, value)
            prob = len(subDataSet) / float(len(dataSet))
            newEntropy += prob * calcShannonEnt(subDataSet)

        infoGain = baseEntropy - newEntropy

        if infoGain > bestInfoGain:
            bestInfoGain = infoGain
            bestFeature = i

    return bestFeature
```

Classification Function

python

```python
def classify(N, d):
    if N.is_leaf():
        return N.class_label
    else:
        split_feature = N.split_feature
        feature_value = d[split_feature]
        child_node = N.children[feature_value]
        return classify(child_node, d)
```

---

7. Overfitting and Pruning

Overfitting Problem

- **Definition**: Model performs well on training data but poorly on test data

- **Causes**:
  - Too many branches
  - Noise in training data
  - Outliers
  - Insufficient training data

Pruning Strategies

1. Pre-pruning (Early Stopping)

- **Concept**: Stop tree construction early
- **Criteria**:
  - Minimum number of samples per node
  - Maximum tree depth
  - Minimum information gain threshold
  - Minimum impurity decrease

Pre-pruning Implementation

python

```python
def chooseBestMultiSplit(dataSet, ops=(0.1, 20)):
    tolG = ops[0]  # Minimum gain threshold
    tolN = ops[1]  # Minimum samples threshold

    # Check minimum samples
    if len(dataSet) < tolN:
        return None

    # ... calculate best split ...

    # Check minimum gain
    if bestInfoGain < tolG:
        return None

    return bestFeature
```

2. Post-pruning

- **Concept**: Build full tree, then prune back

- **Process**:
  1. Build complete tree
  2. Remove branches that don't improve validation accuracy
  3. Use separate validation set for pruning decisions

Stopping Criteria

1. **No more attributes** for splitting
2. **All tuples** share same class label
3. **Empty dataset** (no tuples)
4. **Pre-pruning conditions** met

---

8. Random Forest

Ensemble Methods Overview

- **Definition**: Combine multiple models for better performance
- **Types**:
  - **Bagging**: Average predictions
  - **Boosting**: Weighted voting
  - **Ensemble**: Combine heterogeneous classifiers

Random Forest Concept

- **Definition**: Ensemble of decision trees with randomization
- **Key Features**:
  - Multiple decision trees
  - Random sampling of data and features
  - Voting for final prediction
  - Reduced overfitting

Advantages of Random Forest

1. **Higher Accuracy**: Misclassification only when majority of trees are wrong
2. **Handles Large Data**: Can process large datasets
3. **Parallel Processing**: Trees can be built independently
4. **Robust to Outliers**: Individual tree errors averaged out

Randomization Techniques

1. Bagging (Bootstrap Aggregating)

- **Process**:
  1. For each tree i, create training set Di by sampling with replacement
  2. Build tree Mi on Di
  3. For prediction, each tree votes
  4. Final prediction = majority vote

2. Forest-RI (Random Input Selection)

- **Process**:
  - At each node, randomly select F attributes (F << total attributes)
  - Choose best split among these F attributes
  - Useful for high-dimensional data

3. Forest-RC (Random Linear Combinations)

- **Process**:
  - Create new attributes as linear combinations of existing ones
  - Coefficients are random numbers in [-1, 1]
  - Useful for small or large attribute sets

Random Forest Algorithm

python

```python
def random_forest(D, k, F):
    forest = []

    for i in range(k):
        # Bagging: sample with replacement
        Di = bootstrap_sample(D)

        # Build tree with random feature selection
        tree_i = build_tree_with_random_features(Di, F)
        forest.append(tree_i)

    return forest

def predict(forest, x):
    votes = []
    for tree in forest:
        prediction = tree.predict(x)
        votes.append(prediction)

    # Majority voting
    return majority_vote(votes)
```

---

9. Summary

Decision Tree Advantages

1. **No Domain Knowledge Required**: Algorithm learns patterns automatically
2. **Multidimensional Data**: Handles multiple attributes naturally
3. **Interpretable**: Easy to understand and explain
4. **Fast**: Simple learning and classification
5. **Good Accuracy**: Generally performs well

Decision Tree Limitations

1. **Overfitting**: Can create overly complex trees
2. **Bias**: Different measures have different biases
3. **Instability**: Small data changes can create different trees
4. **Difficulty with Linear Relationships**: May need deep trees

Random Forest Advantages

1. **Reduced Overfitting**: Ensemble reduces individual tree overfitting
2. **Better Accuracy**: Generally more accurate than single trees
3. **Handles Missing Values**: Robust to missing data
4. **Feature Importance**: Provides feature importance rankings
5. **Scalability**: Can handle large datasets

Key Takeaways

- **Decision Trees**: Powerful, interpretable classification method
- **Impurity Measures**: Critical for good split selection
- **Pruning**: Essential for preventing overfitting
- **Random Forest**: Ensemble method that improves upon single trees
- **Implementation**: Recursive algorithms with careful data structure design

---

Practical Considerations

When to Use Decision Trees

- Need interpretable model
- Mixed data types (categorical and continuous)
- Non-linear relationships
- Feature interactions important

When to Use Random Forest

- Accuracy more important than interpretability
- Large datasets
- High-dimensional data
- Want robust predictions

Implementation Tips

1. **Data Preprocessing**: Handle missing values and outliers
2. **Feature Selection**: Remove irrelevant features
3. **Parameter Tuning**: Optimize tree depth, minimum samples, etc.
4. **Validation**: Use cross-validation for model assessment

5. **Ensemble Size**: Balance accuracy vs. computational cost