Data Mining

Classification

- Part 1 -
Outline

1. What is Classification?
2. K-Nearest-Neighbors
3. Decision Trees
4. Model Evaluation
5. Rule Learning
6. Naïve Bayes
7. Support Vector Machines
8. Artificial Neural Networks
9. Hyperparameter Selection
1. What is Classification?

- Goal: Previously unseen records should be assigned a class from a given set of classes as accurately as possible.

- Approach:
  - Given a collection of records (*training set*)
    - each record contains a set of *attributes*
    - one of the attributes is the *class (label)* that should be predicted
  - Learn a *model* for the class attribute as a function of the values of other attributes

- Variants:
  - Single-class problems (class labels e.g. true/false or fraud/no fraud)
  - Multi-class problems (class labels e.g. low, medium, high)
  - Multi-label problems (more than one class per record, e.g. user interests)
Introduction to Classification

A Couple of Questions:
- What is this?
- Why do you know?
- How have you come to that knowledge?
Introduction to Classification

- Goal: Learn a model for recognizing a concept, e.g. trees
- Training data:

  "tree"
  "not a tree"
  "not a tree"
  "tree"
  "not a tree"
  "not a tree"
We (or the learning algorithm) look at positive and negative examples (training data).

... and derive a model.

e.g., "Trees are big, green plants that have a trunk."

Goal: Classification of unseen instances.

Warning: Models are only approximating examples! Not guaranteed to be correct or complete!
Model Learning and Model Application Process

Class/Label Attribute

Training Set

Unseen Records

Learning algorithm

Induction

Apply Model

Deduction

Learn Model

Model
Classification Examples

- **Credit Risk Assessment**
  - Attributes: your age, income, debts, …
  - Class: Are you getting credit by your bank?

- **Marketing**
  - Attributes: previously bought products, browsing behavior
  - Class: Are you a target customer for a new product?

- **Tax Fraud**
  - Attributes: the values in your tax declaration
  - Class: Are you trying to cheat?

- **SPAM Detection**
  - Attributes: words and header fields of an e-mail
  - Class: Is it a spam e-mail?
Classification Techniques

1. K-Nearest-Neighbors
2. Decision Trees
3. Rule Learning
4. Naïve Bayes
5. Support Vector Machines
6. Artificial Neural Networks
7. Deep Neural Networks
8. Many others …
2. K-Nearest-Neighbors

Example Problem

- Predict what the current weather is in a certain place
- where there is no weather station.
- How could you do that?
Basic Idea

- Use the **average of the nearest stations**
- Example:
  - 3x sunny
  - 2x cloudy
  - result = sunny

- This approach is called **K-Nearest-Neighbors**
  - where k is the number of neighbors to consider
  - in the example: k=5
  - in the example: “near” denotes geographical proximity
K-Nearest-Neighbors Classifiers

- Require three things
  - The set of stored records
  - A distance measure to compute distance between records
  - The value of $k$, the number of nearest neighbors to consider

- To classify an unknown record:
  1. Compute distance to each training record
  2. Identify $k$-nearest neighbors
  3. Use class labels of nearest neighbors to determine the class label of unknown record
     - by taking majority vote or
     - by weighing the vote according to distance
Examples of K-Nearest Neighbors

The k-nearest neighbors of a record $x$ are data points that have the $k$ smallest distances to $x$.

(a) 1-nearest neighbor  
(b) 2-nearest neighbor  
(c) 3-nearest neighbor
Choosing a Good Value for K

- If k is too small, the result is sensitive to noise points
- If k is too large, the neighborhood may include points from other classes

- Rule of thumb: Test k values between 1 and 20.
Discussion of K-Nearest-Neighbor Classification

- Often very accurate
  - for instance for optical character recognition (OCR)

- … but slow
  - as training data needs to be searched

- Assumes that all attributes are equally important
  - remedy: attribute selection or attribute weights

- Can handle decision boundaries which are not parallel to the axes (unlike decision trees)
Decision Boundaries of a 1-NN Classifier
KNN Classification in RapidMiner and Python

RapidMiner

Python

```python
from sklearn.neighbors import KNeighborsClassifier

# Train classifier
knn_estimator = KNeighborsClassifier(n_neighbors=3)
knn_estimator.fit(preprocessed_training_data, training_labels)

# Use classifier to predict labels
prediction = knn_estimator.predict(preprocessed_unseen_data)
```
### Resulting Dataset

<table>
<thead>
<tr>
<th>Row No.</th>
<th>Play</th>
<th>prediction(Play)</th>
<th>confidence(no)</th>
<th>confidence(yes)</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>yes</td>
<td>yes</td>
<td>0.451</td>
<td>0.549</td>
<td>sunny</td>
<td>85</td>
<td>85</td>
<td>true</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td>no</td>
<td>0.630</td>
<td>0.370</td>
<td>overcast</td>
<td>80</td>
<td>90</td>
<td>true</td>
</tr>
<tr>
<td>3</td>
<td>yes</td>
<td>yes</td>
<td>0.190</td>
<td>0.810</td>
<td>overcast</td>
<td>83</td>
<td>78</td>
<td>false</td>
</tr>
<tr>
<td>4</td>
<td>yes</td>
<td>no</td>
<td>0.544</td>
<td>0.456</td>
<td>rain</td>
<td>70</td>
<td>95</td>
<td>false</td>
</tr>
<tr>
<td>5</td>
<td>yes</td>
<td>no</td>
<td>0.394</td>
<td>0.606</td>
<td>rain</td>
<td>68</td>
<td>80</td>
<td>true</td>
</tr>
<tr>
<td>6</td>
<td>no</td>
<td>yes</td>
<td>0.250</td>
<td>0.750</td>
<td>rain</td>
<td>65</td>
<td>70</td>
<td>true</td>
</tr>
<tr>
<td>7</td>
<td>yes</td>
<td>yes</td>
<td>0.218</td>
<td>0.782</td>
<td>overcast</td>
<td>64</td>
<td>65</td>
<td>true</td>
</tr>
<tr>
<td>8</td>
<td>no</td>
<td>no</td>
<td>0.559</td>
<td>0.441</td>
<td>sunny</td>
<td>72</td>
<td>95</td>
<td>false</td>
</tr>
<tr>
<td>9</td>
<td>yes</td>
<td>yes</td>
<td>0.212</td>
<td>0.788</td>
<td>sunny</td>
<td>69</td>
<td>70</td>
<td>false</td>
</tr>
<tr>
<td>10</td>
<td>no</td>
<td>yes</td>
<td>0.213</td>
<td>0.787</td>
<td>sunny</td>
<td>75</td>
<td>80</td>
<td>false</td>
</tr>
<tr>
<td>11</td>
<td>yes</td>
<td>yes</td>
<td>0.222</td>
<td>0.778</td>
<td>sunny</td>
<td>68</td>
<td>70</td>
<td>true</td>
</tr>
<tr>
<td>12</td>
<td>yes</td>
<td>no</td>
<td>0.554</td>
<td>0.446</td>
<td>overcast</td>
<td>72</td>
<td>90</td>
<td>true</td>
</tr>
<tr>
<td>13</td>
<td>no</td>
<td>yes</td>
<td>0.164</td>
<td>0.836</td>
<td>overcast</td>
<td>81</td>
<td>75</td>
<td>true</td>
</tr>
<tr>
<td>14</td>
<td>yes</td>
<td>yes</td>
<td>0.250</td>
<td>0.750</td>
<td>rain</td>
<td>71</td>
<td>80</td>
<td>true</td>
</tr>
</tbody>
</table>

- **Prediction**
- **Confidence scores**
Lazy versus Eager Learning

– Lazy Learning
  • Instance-based learning approaches, like KNN, are also called lazy learning as no explicit knowledge (model) is learned
  • **Single goal**: Classify unseen records as accurately as possible

– Eager Learning
  • but actually, we might have **two goals**
    1. classify unseen records
    2. understand the application domain as a human
  • Eager learning approaches generate models that are (might be) interpretable by humans
  • Examples of eager techniques: Decision Tree Learning, Rule Learning
3. Decision Tree Classifiers

Decision trees encode a procedure for taking a classification decision.
Applying a Decision Tree to Unseen Data

Start from the root of tree.

Unseen Record

<table>
<thead>
<tr>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Married</td>
<td>80K</td>
<td>?</td>
</tr>
</tbody>
</table>

Assign Cheat to “No”
The decision boundaries are parallel to the axes because the test condition involves a single attribute at-a-time.
Learning a Decision Tree

- **How to learn a decision tree from training data?**
  - Finding an optimal decision tree is NP-hard
  - Tree building algorithms thus use a greedy, top-down, recursive partitioning strategy to induce a reasonable solution

- **Many different algorithms have been proposed:**
  - Hunt’s Algorithm
  - ID3
  - C4.5
  - CHAID
Hunt’s Algorithm

- Let $D_t$ be the set of training records that reach a node $t$

- Generate leaf node or attribute test:
  - If $D_t$ only contains records that belong to the same class $y_t$, then $t$ is a leaf node labeled as $y_t$
  - If $D_t$ contains records that belong to more than one class, use an attribute test to split the data into subsets having a higher purity.
    - for all possible tests: calculate purity of the resulting subsets
    - choose test resulting in highest purity

- Recursively apply this procedure to each subset.
Hunt’s Algorithm – Step 1

1. We calculate the purity of the resulting subsets for all possible splits
   - Purity of split on Refund
   - Purity of split on Marital Status
   - Purity of split on Taxable Income

2. We find the split on Refund to produce the purest subsets
Hunt’s Algorithm – Step 2

1. We further examine the Refund=No records
2. Again, we test all possible splits
3. We find the split on Marital Status to produce the purest subsets
Hunt’s Algorithm – Step 3

1. We further examine the Marital Status=Single or =Divorced records
2. We find a split on Taxable Income to produce pure subsets
3. We stop splitting as no sets containing different classes are left

<table>
<thead>
<tr>
<th>Tid</th>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>No</td>
<td>Single</td>
<td>70K</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>Divorced</td>
<td>95K</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>Single</td>
<td>85K</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Single</td>
<td>90K</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Refund=No Marital = Single, Divorced Records

All Training Records

Refund

Yes

No

Don’t Cheat

??

Marital Status

Single, Divorced

Married

Taxable Income

< 80K

>= 80K

Don’t Cheat

Cheat

Don’t Cheat

??
1. How should training records be split?
   - How to specify the attribute test condition?
     - Depends on number of ways to split: 2-way split, multi-way split
     - Depends on attribute data type: nominal, ordinal, continuous.
   - How to determine the best split?
     - Different purity measures can be used

2. When should the splitting procedure stop?
   - Shallow trees might generalize better to unseen records
   - Fully grown trees might overfit training data
Splitting Based on Nominal Attributes

- **Multi-way split:** Use as many partitions as distinct values.

  ![Diagram](image)

- **Binary split:** Divides values into two subsets.

  ![Diagram](image)

  {Sports, Luxury} \{Family\} OR {Family, Luxury} \{Sports\}
- **Multi-way split**: Use as many partitions as distinct values.

- **Binary split**: Divides values into two subsets while keeping the order.
### Splitting Based on Continuous Attributes

- **Different ways of handling continuous attributes**

  - **Discretization** to form an ordinal categorical attribute
    - equal-interval binning
    - equal-frequency binning
    - binning based on user-provided boundaries

  - **Binary Decision**: $(A < v)$ or $(A \geq v)$
    - usually sufficient in practice
    - find the best splitting border $v$ based on a purity measure (see below)
    - can be compute intensive

![Binary and multi-way split diagram]

- (ii) Multi-way split
  - Taxable Income?
    - $< 10K$
    - $[10K, 25K]$
    - $[25K, 50K]$
    - $[50K, 80K]$
    - $> 80K$

- (i) Binary split
  - Taxable Income $> 80K$?
    - Yes
    - No
Discretization Example

- Values of the attribute, e.g., age of a person:
  - 0, 4, 12, 16, 16, 18, 24, 26, 28

- Equal-interval binning – for bin width of e.g., 10:
  - Bin 1: 0, 4 \([-,10)\) bin
  - Bin 2: 12, 16, 16, 18 \([10,20)\) bin
  - Bin 3: 24, 26, 28 \([20,+)\) bin
    - denote negative infinity, + positive infinity

- Equal-frequency binning – for bin density of e.g., 3:
  - Bin 1: 0, 4, 12 \([-,14)\) bin
  - Bin 2: 16, 16, 18 \([14,21]\) bin
  - Bin 3: 24, 26, 28 \([21,+]\) bin
3.2 How to Find the Best Split?

Before splitting the dataset contains:
- 10 records of class C0 and
- 10 records of class C1

Which attribute test is the best?
How to Find the Best Split?

- Greedy approach: Test all possible splits and use the one that results in the most **homogeneous (= pure)** nodes.
- Need a measure of **node impurity**:
  - Non-homogeneous
    - High degree of node impurity
  - Homogeneous
    - Low degree of node impurity

- Common measures of node impurity:
  1. GINI Index
  2. Entropy
How to Find the Best Split?

1. Compute impurity measure (P) before splitting

2. Compute impurity measure (M) after splitting
   • Compute impurity measure of each child node
   • M is the weighted impurity of children

3. Choose the attribute test condition that produces the highest purity gain

   Gain = P – M

   or equivalently, lowest impurity measure after splitting (M)
Comparing Two Splits by Purity Gain

Before Splitting:

<table>
<thead>
<tr>
<th>Node</th>
<th>C0</th>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A? Yes  
Node N1

A? No  
Node N2

B? Yes  
Node N3

B? No  
Node N4

Higher purity gain?  \( P - M_{12} \) or \( P - M_{34} \)
3.2.1 Impurity Measure: GINI Index

- GINI Index for a given node $t$:

$$GI_{NI}(t) = 1 - \sum_{j} [p(j \mid t)]^2$$

- $p(j \mid t)$ is the relative frequency of class $j$ at node $t$.

- Minimum (0.0) when all records belong to one class.

- Maximum (1 - $1/n_c$) when records are equally distributed among all classes. $n_c = \text{number of classes}$.
Examples for computing GINI

\[ GINI(t) = 1 - \sum_j [p(j | t)]^2 \]

<table>
<thead>
<tr>
<th>C1</th>
<th>0</th>
<th>P(C1) = 0/6 = 0</th>
<th>P(C2) = 6/6 = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td>6</td>
<td>Gini = 1 – P(C1)^2 – P(C2)^2 = 1 – 0 – 1 = 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C1</th>
<th>1</th>
<th>P(C1) = 1/6</th>
<th>P(C2) = 5/6</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td>5</td>
<td>Gini = 1 – (1/6)^2 – (5/6)^2 = 0.278</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C1</th>
<th>2</th>
<th>P(C1) = 2/6</th>
<th>P(C2) = 4/6</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td>4</td>
<td>Gini = 1 – (2/6)^2 – (4/6)^2 = 0.444</td>
<td></td>
</tr>
</tbody>
</table>
Splitting Based on GINI

- When a node $p$ is split into $k$ partitions (subsets), the GINI index of each partition is weighted according to the partition's size.

- The quality of the overall split is computed as:

$$GINI_{\text{split}} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

where: $n_i = \text{number of records at child } i$

$n = \text{number of records at node } p$
Example: Calculating the Purity Gain of a Split

Split into two partitions

\[ \text{GINI}_{N1} = 1 - \left(\frac{5}{7}\right)^2 - \left(\frac{2}{7}\right)^2 = 0.408 \]

\[ \text{GINI}_{N2} = 1 - \left(\frac{1}{5}\right)^2 - \left(\frac{4}{5}\right)^2 = 0.32 \]

\[ \text{GINI}_{\text{Split}} = \frac{7}{12} \times 0.408 + \frac{5}{12} \times 0.32 = 0.371 \]

Purity Gain = 0.5 – 0.371 = 0.129

<table>
<thead>
<tr>
<th>Parent</th>
<th>N1</th>
<th>N2</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>C2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Weights
Categorical Attributes: Computing Gini Index

For each distinct attribute value, gather counts for each class.

### Multi-way split

<table>
<thead>
<tr>
<th>CarType</th>
<th>Family</th>
<th>Sports</th>
<th>Luxury</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>C2</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Gini**: 0.393

### Two-way split

**Find best partition of values**

#### Case 1: CarType = \{Sports, Luxury\}

<table>
<thead>
<tr>
<th>CarType</th>
<th>{Sports, Luxury}</th>
<th>{Family}</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>C2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

- **Gini**: 0.400

#### Case 2: CarType = \{Sports\}, \{Family, Luxury\}

<table>
<thead>
<tr>
<th>CarType</th>
<th>{Sports}</th>
<th>{Family, Luxury}</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

- **Gini**: 0.419
Continuous Attributes: Computing Gini Index

- How to find the best binary split for a continuous attribute?
- Efficient computation:
  1. Sort the attribute on values
  2. Linearly scan these values, each time updating the count matrix and computing the gini index
  3. Choose the split position that has the smallest gini index

<table>
<thead>
<tr>
<th>Taxable Income</th>
<th>60</th>
<th>70</th>
<th>75</th>
<th>80</th>
<th>85</th>
<th>90</th>
<th>95</th>
<th>100</th>
<th>110</th>
<th>120</th>
<th>125</th>
<th>172</th>
<th>220</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;</td>
</tr>
<tr>
<td>Yes</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>No</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Gini</td>
<td>0.420</td>
<td>0.400</td>
<td>0.375</td>
<td>0.343</td>
<td>0.417</td>
<td>0.400</td>
<td>0.300</td>
<td>0.400</td>
<td>0.343</td>
<td>0.375</td>
<td>0.400</td>
<td>0.420</td>
<td></td>
</tr>
</tbody>
</table>
3.2.2 Alternative Impurity Measure: Information Gain

- Information gain relies on the entropy of each node
- Entropy of a given node t:

$$\text{Entropy} \ (t) = - \sum_j p(j \mid t) \log_2 p(j \mid t)$$

$p(j \mid t)$ is the relative frequency of class j at node t

- Entropy measures homogeneity of a node
  - Minimum (0.0) when all records belong to one class
  - Maximum (log $n_c$) when records are equally distributed among all classes
Examples for Computing Entropy

\[
Entropy (t) = - \sum_j p(j | t) \log_2 p(j | t)
\]

| C1 | 0 | P(C1) = 0/6 = 0  \quad P(C2) = 6/6 = 1  
| C2 | 6 | Entropy = \(- 0 \log_2 0 - 1 \log_2 1\) = \(- 0 - 0\) = 0 |

| C1 | 1 | P(C1) = 1/6  \quad P(C2) = 5/6  
| C2 | 5 | Entropy = \(- (1/6) \log_2 (1/6) - (5/6) \log_2 (5/6)\) = 0.65 |

| C1 | 2 | P(C1) = 2/6  \quad P(C2) = 4/6  
| C2 | 4 | Entropy = \(- (2/6) \log_2 (2/6) - (4/6) \log_2 (4/6)\) = 0.92 |
Splitting Based on Information Gain

- Information Gain:

\[ GAIN_{\text{split}} = Entropy(p) - \left( \sum_{i=1}^{k} \frac{n_i}{n} \cdot Entropy(i) \right) \]

Parent Node \( p \) is split into \( k \) partitions;
\( n_i \) is number of records in partition \( i \)

- Information gain measures the entropy reduction of a split.
- We choose the split with the largest reduction (maximal GAIN)
- Disadvantage: Tends to prefer splits that result in large number of partitions, each being small but pure (split by ID attribute?)
3.2.3 Alternative Splitting Criterion: GainRATIO

- GainRATIO is designed to overcome the tendency to generate a large number of small partitions.
- GainRATIO adjusts information gain by the entropy of the partitioning (SplitINFO).
- Higher entropy of the partitioning (large number of small partitions) is penalized!

\[
\text{GainRATIO}_{\text{split}} = \frac{GAIN_{\text{split}}}{\text{SplitINFO}}
\]

\[
\text{SplitINFO} = -\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}
\]

Parent Node p is split into k partitions
\(n_i\) is the number of records in partition i
3.3 Overfitting

- We want to learn models that are good at classifying unseen records
- **Overfitting**: Learned models can fit the training data too closely and thus work poorly on unseen data
- Model perfectly fitting the training data:
  "Trees are **big, green plants** that have a **trunk** and **no wheels**"
- Unseen example:

- **Goal**: Find good compromise between specificity and generality of the learned model
Overfitting: Second Example

- Example: Predict credit rating
  - possible decision tree:

<table>
<thead>
<tr>
<th>Name</th>
<th>Net Income</th>
<th>Job status</th>
<th>Debts</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>40000</td>
<td>employed</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>Mary</td>
<td>38000</td>
<td>employed</td>
<td>10000</td>
<td>-</td>
</tr>
<tr>
<td>Stephen</td>
<td>21000</td>
<td>self-employed</td>
<td>20000</td>
<td>-</td>
</tr>
<tr>
<td>Eric</td>
<td>2000</td>
<td>student</td>
<td>10000</td>
<td>-</td>
</tr>
<tr>
<td>Alice</td>
<td>35000</td>
<td>employed</td>
<td>4000</td>
<td>+</td>
</tr>
</tbody>
</table>
Overfitting: Second Example

- Example: Predict credit rating
  - alternative decision tree:

<table>
<thead>
<tr>
<th>Name</th>
<th>Net Income</th>
<th>Job status</th>
<th>Debts</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>40000</td>
<td>employed</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>Mary</td>
<td>38000</td>
<td>employed</td>
<td>10000</td>
<td>-</td>
</tr>
<tr>
<td>Stephen</td>
<td>21000</td>
<td>self-employed</td>
<td>20000</td>
<td>-</td>
</tr>
<tr>
<td>Eric</td>
<td>2000</td>
<td>student</td>
<td>10000</td>
<td>-</td>
</tr>
<tr>
<td>Alice</td>
<td>35000</td>
<td>employed</td>
<td>4000</td>
<td>+</td>
</tr>
</tbody>
</table>
Overfitting: Second Example

- Both trees seem equally good
  - as they classify all instances in the training set correctly

- Which one do you prefer?

```
Name = "John"

No

Name= "Alice"

Yes

-  +

Debts >5000

Yes

No

-  +
```
Occam's Razor

- Named after William of Ockham (1287-1347)
- A fundamental principle of science
  - if you have two theories
  - that explain a phenomenon equally well
  - choose the simpler one
- Tree that likely generalizes better according to Occam’s razor

![Occam's Razor Diagram](image)

Diagram:
- Debts >5000
  - Yes
    - 
  - No
    +
Overfitting: Symptoms and Causes

- Symptoms:
  1. decision tree too deep
  2. too many branches
  3. model works well on training set but performs bad on test set

- Typical causes of overfitting
  1. too little training data
  2. noise / outliers
  3. poor learning algorithm

An overfitted model does not generalize well to unseen data.
Example of an Outlier causing Overfitting

(A) A partition of the data space

(B) The decision tree

Outlier modelled by subtree

Simpler tree likely generalizes better
Underfitting versus Overfitting

Underfitting: when model is too simple, both training and test errors are large
Overfitting: when model is too complex, training error is small but test error is large
How to Prevent Overfitting 1: Use More Training Data

- If training data is under-representative, testing errors increase and training errors decrease on increasing number of nodes.
- Increasing the size of training data reduces the difference between training and testing errors at a given number of nodes.
How to Prevent Overfitting 2: Pre-Pruning

1. Stop the algorithm before tree becomes fully-grown
   • shallower tree potentially generalizes better

2. Normal stopping conditions for a node (no pruning):
   • Stop if all instances belong to the same class
   • Stop if all the attribute values are the same

3. Early stopping conditions (pre-pruning):
   • Stop if number of instances within a leaf node is less than some user-specified threshold (e.g. leaf size < 4)
   • Stop if expanding the current node only slightly improves the impurity measure (e.g. gain < 0.01)
How to Prevent Overfitting 3: Post-Pruning

1. Grow decision tree to its entirety
2. trim the nodes of the decision tree in a bottom-up fashion
3. Estimate generalization error before and after trimming
   - using a validation set
4. If generalization error improves after trimming
   - replace sub-tree by a leaf node or
   - replace subtree by most frequently used branch
How to Prevent Overfitting 4: Ensembles

- Lean different models (base learners)
- Have them vote on the final classification decision

- Idea: Wisdom of the crowds applied to classification
  - A single classifier might focus too much on one aspect
  - Multiple classifiers can focus on different aspects
Random Forest

- Ensemble consisting of a large number of different decision trees

- Independence of trees achieved by introducing randomness into the learning process
  - only use a random subset of the attributes at each split
  - learn on different random subsets of the data (bagging)

- Random forests usually outperform single decision trees
Decision Tree Classification in RapidMiner and Python

RapidMiner

Python

```python
from sklearn.tree import DecisionTreeClassifier

# Train classifier
dt_learner = DecisionTreeClassifier(criterion='gini', max_depth=10)
dt_learner.fit(preprocessed_training_data, training_labels)

# Use classifier to predict labels
prediction = dt_learner.predict(preprocessed_unseen_data)
```
Examples of Learned Decision Trees

**Python**

- Petal width (cm) ≤ 0.8
  - Gini = 0.665
  - Samples = 100
  - Value = [33, 31, 36]

True

- Gini = 0.0
  - Samples = 33
  - Value = [33, 0, 0]

- Petal width (cm) ≤ 1.65
  - Gini = 0.497
  - Samples = 67
  - Value = [0, 31, 36]

False

- Gini = 0.165
  - Samples = 33
  - Value = [0, 30, 3]

- Gini = 0.057
  - Samples = 34
  - Value = [0, 1, 33]
Random Forests in RapidMiner and Python

**RapidMiner**

**Python**

```python
from sklearn.ensemble import RandomForestClassifier

# Train classifier
forest_estimator = RandomForestClassifier(n_estimators=100, criterion='gini', max_depth=None)
forest_estimator.fit(preprocessed_training_data, training_labels)

# Use classifier to predict labels
prediction = forest_estimator.predict(preprocessed_unseen_data)
```
3.4 Discussion of Decision Trees

- **Advantages**
  - Inexpensive to construct
  - Extremely fast at classifying unknown records
  - Easy to interpret by humans for small-sized trees (eager learning)
  - Can easily handle redundant or irrelevant attributes
  - Accuracy is comparable to other classification techniques for many low dimensional data sets

- **Disadvantages**
  - Space of possible decision trees is exponentially large. Greedy approaches are often unable to find the best tree
  - Does not take into account interactions between attributes

Chapter 3: Classification

Chapter 6.3: Nearest Neighbor Classifiers

Chapter 3.3: Decision Tree Classifier

Chapter 3.4: Overfitting

Chapter 6.10.6: Random Forests