Recap

- **k-Nearest-Neighbours** as a lazy classification method that classifies a previously unseen data point based on the classes of the $k$ closest **data points** from the training data.

- **Naïve Bayes** as a probabilistic classification method that is primarily used for **text classification**, but can be extended to deal with other data.
3.5 Decision Trees

- **Decision trees** are a family of classification methods that support an arbitrary number of classes.

- A decision tree provides a sequence of (binary) decisions which have to be made in order to decide which class a data point belongs to.

- Decision trees are often praised for their interpretability: we can see why a data point ended up in a specific class, and—in principle—the classification could be performed by a human user not knowing anything about ML.
Decision Trees

- Weight [lbs] \( \leq 2959.5 \)
  - entropy = 1.3301
  - samples = 313
  - value = [195, 53, 65]
  - class = U.S.A.

- Weight [lbs] \( > 2959.5 \)
  - entropy = 1.5629
  - samples = 174
  - value = [65, 44, 65]
  - class = U.S.A.

- True
  - entropy = 1.2667
    - samples = 37
    - value = [3, 12, 22]
    - class = Japan

- False
  - Power [HP] \( \leq 134.0 \)
    - entropy = 0.346
      - samples = 139
      - value = [130, 9, 0]
      - class = U.S.A.

  - entropy = 0.5436
    - samples = 72
    - value = [63, 9, 0]
    - class = U.S.A.

  - entropy = 0.0
    - samples = 67
    - value = [67, 0, 0]
    - class = U.S.A.
Information Theory

- Claude E. Shannon (1916-2001) proposed **Information Theory**, which is important for, e.g.:
  - **encoding** and **compression**
  - **data transmission** (e.g., in networks)

  and has **applications** in

  - **Machine Learning**
  - **Information Retrieval**
  - **Natural Language Processing**
Entropy

- **Entropy** quantifies the amount of uncertainty of a random variable $Y$

$$H(Y) = - \sum_y P[y] \log_2 P[y]$$

- **Example**: Let’s consider a fair coin

$$P(Y = \text{head}) = P(Y = \text{tail}) = \frac{1}{2}$$

$$H(Y) = - \left( \frac{1}{2} \log_\frac{1}{2} + \frac{1}{2} \log \frac{1}{2} \right) = 1$$
Entropy

- **Entropy** quantifies the *amount of uncertainty* of a random variable $Y$

  $$H(Y) = - \sum_y P[y] \log_2 P[y]$$

- **Example**: Let’s consider a **biased coin**

  $$P(Y = \text{head}) = \frac{3}{4} \quad P(Y = \text{tail}) = \frac{1}{4}$$

  $$H(Y) = - \left( \frac{3}{4} \log_\frac{3}{4} + \frac{1}{4} \log_\frac{1}{4} \right) \approx 0.6579$$

- **Less uncertainty** for biased coin, or put differently, knowing outcome of **fair coin** provides more information
**Entropy**

- **Entropy** quantifies the **amount of uncertainty** of a random variable $Y$

$$
H(Y) = - \sum_{y} P[y] \log_2 P[y]
$$

- **Example**: Let’s consider a **fair dice**

$$
P(Y = n) = \frac{1}{6} \quad \text{with} \quad n \in \{1, \ldots, 6\}
$$

$$
H(Y) = - \left( \frac{1}{6} \log \frac{1}{6} + \ldots + \frac{1}{6} \log \frac{1}{6} \right) \approx 2.5849
$$
Entropy

- **Entropy** quantifies the **amount of uncertainty** of a random variable $Y$

  $$H(Y) = - \sum_y P[y] \log_2 P[y]$$

- **Example**: Let’s consider a **biased dice**

  $$P(Y = 1) = \frac{1}{2}$$

  $$P(Y = n) = \frac{1}{10} \quad \text{with} \quad n \in \{2, \ldots, 6\}$$

  $$H(Y) = - \left( \frac{1}{2} \log \frac{1}{2} + \frac{1}{10} \log \frac{1}{10} + \ldots + \frac{1}{10} \log \frac{1}{10} \right) \approx 2.1609$$
Huffman Coding

- **Entropy** provides a **lower bound** on the **average number of bits** (code length) required to encode/transmit the value of the random variable.

- **Huffman coding** assigns variable-length code words to values of the random variable in almost **optimal manner** (i.e., achieving almost a minimum-length representation).

- **Idea:** Assign **shorter code words** to values that occur more frequently, i.e., have higher probability.
Huffman Coding

- **Example**: Huffman coding for our **biased coin**

<table>
<thead>
<tr>
<th>$n$</th>
<th>$P[Y = n]$</th>
<th>Code Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1/10</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>1/10</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>1/10</td>
<td>1110</td>
</tr>
<tr>
<td>5</td>
<td>1/10</td>
<td>11110</td>
</tr>
<tr>
<td>6</td>
<td>1/10</td>
<td>11111</td>
</tr>
</tbody>
</table>

- Code is **prefix-free**, i.e., no code word is a prefix of another code word (allows interpreting a stream of code words)

- This code achieves an **average code length of 2.4** (which is less than 3 bits for the standard binary encoding)
Conditional Entropy

- **Conditional entropy** quantifies the **amount of uncertainty** of a random variable $Y$ if the value of another random variable $X$ is known

$$H(Y|X) = \sum_x P(x) H(Y|x)$$

- **Example**: Let’s again consider a **fair dice**
  - random variable $Y$ indicates **face of dice** $\{1, \ldots, 6\}$
  - random variable $X$ indicates whether **face** is **odd** or **even** $\{o,e\}$

$$P(Y = n) = \frac{1}{6} \quad \text{with} \quad n \in \{1, \ldots, 6\}$$

$$P(X = o) = P(X = e) = \frac{1}{2}$$
Example (cont’d): Let’s look at the conditional probabilities

<table>
<thead>
<tr>
<th>$y$</th>
<th>$x$</th>
<th>$P[y \mid x]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>o</td>
<td>1/3</td>
</tr>
<tr>
<td>1</td>
<td>e</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>o</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>e</td>
<td>1/3</td>
</tr>
<tr>
<td>3</td>
<td>o</td>
<td>1/3</td>
</tr>
<tr>
<td>3</td>
<td>e</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>o</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>e</td>
<td>1/3</td>
</tr>
<tr>
<td>5</td>
<td>o</td>
<td>1/3</td>
</tr>
<tr>
<td>5</td>
<td>e</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>o</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>e</td>
<td>1/3</td>
</tr>
</tbody>
</table>

$H(Y \mid o) = - \left( \frac{1}{3} \log \frac{1}{3} + 0 + \frac{1}{3} \log \frac{1}{3} + 0 + \ldots \frac{1}{3} \log \frac{1}{3} + 0 \right) 
\approx 1.5849$

$H(Y \mid e) = - \left( 0 + \frac{1}{3} \log \frac{1}{3} + 0 + \frac{1}{3} \log \frac{1}{3} + 0 + \ldots \frac{1}{3} \log \frac{1}{3} \right) 
\approx 1.5849$

$H(Y \mid X) = \frac{1}{2} H(Y \mid o) + \frac{1}{2} H(Y \mid e) 
\approx 1.5849$

Intuition: Knowing whether an odd or even face is shown reduces the amount of uncertainty
Information Gain

- **Information gain** (also: mutual information) measures how much information is gained about a random variable $Y$ if the value of another random variable $X$ is known.

\[
I(X, Y) = \sum_x \sum_y P[x \wedge y] \log \left( \frac{P[x \wedge y]}{P[x]P[y]} \right)
\]

which can be rewritten as (cf. exercise)

\[
I(X, Y) = H(X) - H(X|Y)
\]

\[
I(X, Y) = H(Y) - H(Y|X)
\]
Information Gain

- **Example**: Let’s again consider a **fair dice**
  - random variable $Y$ indicates **face of dice** $\{1,\ldots,6\}$
  - random variable $X$ indicates whether **face** is **odd** or **even** $\{o,e\}$

\[
I(X, Y) = H(Y) - H(Y|X) \\
\approx 2.5849 - 1.5849 = 1
\]
Recursively Growing a Decision Tree

- We’ll now learn to generate a decision tree recursively
  - random variable $Y$ indicates class labels (e.g., Europe, U.S.)
  - random variable $X$ indicates outcome of a binary decision (e.g., does the car weight more than 2000 lbs)

- In each step, we need to determine the best possible decision criterion to split the remaining data points (hint: we’ll use information gain to this end)
Recursively Growing a Decision Tree

- **Recursive top-down generation** of a decision tree (i.e., from root to leaves)
  - if only few data points are left or most of them belong to a single class, then generate a leaf with majority class
  - otherwise, identify the best possible decision criterion, split the data accordingly, and recursively generate left and right decision subtree
Node decisionTree(Data d) {
    // Generate leaf node, if data is pure
    if (isPure(d)) {
        return Leaf(majorityClass(d));
    }

    // Determine best decision criterion
    Condition c = bestSplit(d);

    // Split the data accordingly
    Data ld, rd = split(d, c);

    // Recursively generate subtree
    Node lc = decisionTree(ld);
    Node rc = decisionTree(rd);

    return new Node(c, lc, rc);
}
Splits for Numerical and Ordinal Features

- For **numerical** and **ordinal** features we can compare the value of the feature $x$ against a **threshold** $a$

  $$X : x \leq a$$

- The random variable $X$ thus indicates whether the feature value $x$ of a data point is **smaller or equal** than the threshold $a$ (true) or **larger** (false)

- In theory, there can be an **infinite number of possible** thresholds $a$ to consider; in practice, it is good enough to consider the feature values that **appear in the data**
Splits for Numerical and Ordinal Features

- **Example**: Let’s assume that we have observed the following values for a feature \( x \) in our training data

\[
x \in \{2, 4, 5, 6, 8, 10, 12\}
\]

We need to consider the following decision criteria

\[
X : x \leq a \quad \text{with} \quad a \in \{2, 4, 5, 6, 8, 10\}
\]

- **Note**: Some implementations consider thresholds midway between values observed in the data, i.e.:}

\[
X : x \leq a \quad \text{with} \quad a \in \{3.5, 4.5, 5.5, 7, 9, 11\}
\]
Splits for Nominal Features

- For nominal features we can check whether the value of the feature $x$ is in a **subset** $A$ of observed feature values

  $$X : x \in A$$

- The random variable $X$ thus indicates whether the feature value $x$ of a data point is contained in the **subset** $A$ (true) or not (false)

- If there are $n$ observed values for feature $x$ in our data, we need to consider an **exponential number of** $2^{(n-1)} - 1$ subsets $A$
Splits for Nominal Features

- **Example**: Let’s assume that we have observed the following values for a feature $x$ in our training data

  $$x \in \{r, g, b, w\}$$

  We need to consider the following decision criteria

  $$X : x \in A \quad \text{with}$$

  $$A \in \{\{r\}, \{g\}, \{b\}, \{w\}, \{r, g\}, \{r, b\}, \{r, w\}, \{r, w\}\}$$

- **Note**: There is no need to consider all $2^n$ subsets, because (i) the empty set {} does not split the data and (ii) a subset and its complementary set result in the same split (e.g., $\{r\}$ and $\{g, b, w\}$ in our example)
Pruning a Decision Tree

- In each step, we identify the best possible decision criterion by considering all options across all features, and choosing the one with highest information gain.

- If we grow the decision tree until all leaves contain only data points from a single class, we’re prone to overfitting.

- It is better to prune the decision tree by stopping the recursive generation, once less than minSize data points are seen or at least minPercentage percent of data points belong to the majority class.
Decision Trees (Example)

- **Example**: Predict the **risk** of an insurance customer based on **age** and **type of car**

<table>
<thead>
<tr>
<th>Age</th>
<th>Type of Car</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Roadster</td>
<td>Low</td>
</tr>
<tr>
<td>20</td>
<td>Oldtimer</td>
<td>High</td>
</tr>
<tr>
<td>25</td>
<td>Roadster</td>
<td>Low</td>
</tr>
<tr>
<td>45</td>
<td>SUV</td>
<td>High</td>
</tr>
<tr>
<td>20</td>
<td>Roadster</td>
<td>High</td>
</tr>
<tr>
<td>25</td>
<td>SUV</td>
<td>High</td>
</tr>
</tbody>
</table>

\[
H(Risk) = - \left( \frac{1}{3} \log \frac{1}{3} + \frac{2}{3} \log \frac{2}{3} \right) = 0.9183
\]
Decision Trees (Example)

- **Step 1:** Determine the root of the decision tree
  - Decision criterion \( \text{Age} : 20 \)

\[
H(Risk | \text{Age} : 20) = \frac{1}{3} H(Risk | \text{Age} \leq 20) + \frac{2}{3} H(Risk | \text{Age} > 20)
\]

\[
= \frac{1}{3} (0) + \frac{2}{3} (1)
\]

\[
= 0.6667
\]

\[
I(Risk, \text{Age} : 20) = 0.9183 - 0.6667
\]

\[
= 0.2516
\]

<table>
<thead>
<tr>
<th>Age</th>
<th>Type of Car</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Roadster</td>
<td>Low</td>
</tr>
<tr>
<td>20</td>
<td>Oldtimer</td>
<td>High</td>
</tr>
<tr>
<td>25</td>
<td>Roadster</td>
<td>Low</td>
</tr>
<tr>
<td>45</td>
<td>SUV</td>
<td>High</td>
</tr>
<tr>
<td>20</td>
<td>Roadster</td>
<td>High</td>
</tr>
<tr>
<td>25</td>
<td>SUV</td>
<td>High</td>
</tr>
</tbody>
</table>
Decision Trees (Example)

- Decision criterion \( \text{Age : 25} \)

\[
H(Risk|\text{Age : 25}) = \frac{5}{6} H(Risk|\text{Age} \leq 25) + \frac{1}{6} H(Risk|\text{Age} > 25)
\]

\[
= \frac{5}{6} \left( - \left( \frac{2}{5} \log \frac{2}{5} + \frac{3}{5} \log \frac{3}{5} \right) \right) + \frac{1}{6} (0)
\]

\[
= 0.8091
\]

\[
I(Risk, \text{Age : 25}) = 0.9183 - 0.8091
\]

\[
= 0.1092
\]
Decision Trees (Example)

- Decision criterion Type : \{\text{Oldtimer}\}

\[
H(Risk \mid Type : \{O\}) = \frac{1}{6}H(Risk \mid Type \in \{O\}) + \frac{5}{6}H(Risk \mid Type \notin \{O\})
\]

\[
= \frac{1}{6} (0) + \frac{5}{6} \left( - \left( \frac{2}{5} \log_2 \frac{2}{5} + \frac{3}{5} \log_3 \frac{3}{5} \right) \right)
\]

\[
= 0.8091
\]

\[
I(Risk, Type : \{O\}) = 0.9183 - 0.8091
\]

\[
= 0.1092
\]
Decision Trees (Example)

- **Decision criterion** Type : \{Roadster\}

\[
H(Risk \mid Type : \{R\}) = \frac{3}{6} H(Risk \mid Type \in \{R\}) + \frac{3}{6} H(Risk \mid Type \notin \{R\})
\]

\[
= \frac{3}{6} \left( - \left( \frac{2}{3} \log_2 \frac{2}{3} + \frac{1}{3} \log_2 \frac{1}{3} \right) \right) + \frac{3}{6} (0)
\]

\[
= 0.4591
\]

\[
I(Risk, Type : \{R\}) = 0.9183 - 0.4591
\]

\[
= 0.4592
\]

<table>
<thead>
<tr>
<th>Age</th>
<th>Type of Car</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Roadster</td>
<td>Low</td>
</tr>
<tr>
<td>20</td>
<td>Oldtimer</td>
<td>High</td>
</tr>
<tr>
<td>25</td>
<td>Roadster</td>
<td>Low</td>
</tr>
<tr>
<td>45</td>
<td>SUV</td>
<td>High</td>
</tr>
<tr>
<td>20</td>
<td>Roadster</td>
<td>High</td>
</tr>
<tr>
<td>25</td>
<td>SUV</td>
<td>High</td>
</tr>
</tbody>
</table>
Decision Trees (Example)

- Decision criterion Type : \{SUV\}

\[ H(Risk \mid Type : \{S\}) = \frac{2}{6} H(Risk \mid Type \in \{S\}) + \frac{4}{6} H(Risk \mid Type \notin \{S\}) \]

\[ = \frac{2}{6} (0) + \frac{4}{6} \left( - \left( \frac{1}{2} \log \frac{1}{2} + \frac{1}{2} \log \frac{1}{2} \right) \right) \]

\[ = 0.6667 \]

\[ I(Risk, Type : \{S\}) = 0.9183 - 0.6667 \]

\[ = 0.2516 \]

<table>
<thead>
<tr>
<th>Age</th>
<th>Type of Car</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Roadster</td>
<td>Low</td>
</tr>
<tr>
<td>20</td>
<td>Oldtimer</td>
<td>High</td>
</tr>
<tr>
<td>25</td>
<td>Roadster</td>
<td>Low</td>
</tr>
<tr>
<td>45</td>
<td>SUV</td>
<td>High</td>
</tr>
<tr>
<td>20</td>
<td>Roadster</td>
<td>High</td>
</tr>
<tr>
<td>25</td>
<td>SUV</td>
<td>High</td>
</tr>
</tbody>
</table>
Decision Trees (Example)

- Pick Type: \{Roadster\} as the **decision criterion** for the **root node**, because it achieves the highest information gain.

- **Split** the data accordingly and **recursively grow** decision trees for the obtained subsets of the data.

<table>
<thead>
<tr>
<th>Age</th>
<th>Type of Car</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Roadster</td>
<td>Low</td>
</tr>
<tr>
<td>25</td>
<td>Roadster</td>
<td>Low</td>
</tr>
<tr>
<td>20</td>
<td>Roadster</td>
<td>High</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Age</th>
<th>Type of Car</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Oldtimer</td>
<td>High</td>
</tr>
<tr>
<td>45</td>
<td>SUV</td>
<td>High</td>
</tr>
<tr>
<td>25</td>
<td>SUV</td>
<td>High</td>
</tr>
</tbody>
</table>
Decision Trees (Example)

- For the **right subtree**, there is **nothing to do**, since all data points belong to a **single class**

```
Age | Type of Car | Risk
--- | ------------|------
25  | Roadster    | Low  
25  | Roadster    | Low  
20  | Roadster    | High 
```

```
Age | Type of Car | Risk
--- | ------------|------
20  | Oldtimer    | High 
45  | SUV         | High 
25  | SUV         | High 
```

Type : {Roadster}
Decision Trees (Example)

- For the **left subtree**, there is only one decision criterion to consider, namely \( \text{Age} : 20 \)

\[
H(Risk) = - \left( \frac{1}{3} \log \frac{1}{3} + \frac{2}{3} \log \frac{2}{3} \right) = 0.9183
\]

\[
H(Risk | \text{Age} : 20) = \frac{1}{3} H(Risk | \text{Age} \leq 20) + \frac{2}{3} H(Risk | \text{Age} > 20)
\]

\[
= \frac{1}{3} (0) + \frac{2}{3} (0)
\]

\[
= 0
\]

\[
I(Risk, \text{Age} : 20) = 0.9183 - 0
\]

\[
= 0.9183
\]

<table>
<thead>
<tr>
<th>Age</th>
<th>Type of Car</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Roadster</td>
<td>Low</td>
</tr>
<tr>
<td>25</td>
<td>Roadster</td>
<td>Low</td>
</tr>
<tr>
<td>20</td>
<td>Roadster</td>
<td>High</td>
</tr>
</tbody>
</table>
Decision Trees (Example)

```
<table>
<thead>
<tr>
<th>Age</th>
<th>Type of Car</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Roadster</td>
<td>High</td>
</tr>
<tr>
<td>25</td>
<td>Roadster</td>
<td>Low</td>
</tr>
<tr>
<td>25</td>
<td>Roadster</td>
<td>Low</td>
</tr>
</tbody>
</table>
```
Decision Trees (Example)

Type: {Roadster}

Age: 20

- True: High
- False:
  - True: High
  - False: Low
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

# load data
cars = pd.read_csv("/path-to/auto-mpg.data",sep="\s+",header=None)

# extract power and weight as data matrix X
X = cars.iloc[:, [3,4]].values

# extract origin as target vector y
y = cars.iloc[:, 7].values

# split into training data (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# learn decision tree
tree = DecisionTreeClassifier(criterion='entropy')
tree.fit(X_train,y_train)
y_predicted = tree.predict(X_test)

# compute confusion matrix
print(confusion_matrix(y_true=y_test, y_pred=y_predicted))

# compute accuracy
print(accuracy_score(y_true=y_test, y_pred=y_predicted)) # 0.746835443038
from sklearn.tree import export_graphviz

export_graphviz(tree, out_file='tree.dot', feature_names=['Power [HP]', 'Weight [lbs]'], class_names=['U.S.A.', 'Europe', 'Japan'])
# you need to have Graphviz (graphviz.org) installed to open the generated file
# to generate a PDF from the .dot file, run:
# dot -Tpdf tree.dot -o tree.pdf

- Ouch, can we generate a simpler decision tree?
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

# load data
cars = pd.read_csv("/path-to/auto-mpg.data", sep="\s+", header=None)

# extract power and weight as data matrix X
X = cars.iloc[:, [3, 4]].values

# extract origin as target vector y
y = cars.iloc[:, 7].values

# split into training data (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# learn decision tree of maximal depth 3
tree = DecisionTreeClassifier(criterion='entropy', max_depth=3)
tree.fit(X_train, y_train)
y_predicted = tree.predict(X_test)

# compute confusion matrix
print(confusion_matrix(y_true=y_test, y_pred=y_predicted))

# compute accuracy
print(accuracy_score(y_true=y_test, y_pred=y_predicted))  # 0.632911392405
Plotting the Decision Tree

```
Weight [lbs] <= 2959.5
  entropy = 1.3301
  samples = 313
  value = [195, 53, 65]
  class = U.S.A.
```

True

```
Weight [lbs] <= 2030.0
  entropy = 1.5629
  samples = 174
  value = [65, 44, 65]
  class = U.S.A.
```

```
Power [HP] <= 134.0
  entropy = 0.346
  samples = 139
  value = [130, 9, 0]
  class = U.S.A.
```

False

```
Power [HP] <= 50.5
  entropy = 1.2667
  samples = 37
  value = [3, 12, 22]
  class = Japan
```

```
Power [HP] <= 92.5
  entropy = 1.5324
  samples = 137
  value = [62, 32, 43]
  class = U.S.A.
```

```
Power [HP] <= 80.5
  entropy = 0.5436
  samples = 72
  value = [63, 9, 0]
  class = U.S.A.
```

```
entropy = 0.0
  samples = 4
  value = [0, 4, 0]
  class = Europe
```

```
entropy = 1.2001
  samples = 33
  value = [3, 8, 22]
  class = Japan
```

```
entropy = 1.4503
  samples = 92
  value = [50, 22, 20]
  class = U.S.A.
```

```
entropy = 1.4856
  samples = 45
  value = [12, 10, 23]
  class = Japan
```

```
entropy = 1.0
  samples = 6
  value = [3, 3, 0]
  class = U.S.A.
```

```
entropy = 0.4395
  samples = 66
  value = [60, 6, 0]
  class = U.S.A.
```
3.6 Ensemble Learning

- Ensemble learning determines **multiple classification models** (e.g., single method on different samples or multiple methods) and **aggregates** their predictions.

- Ensembles are often **more robust** than individual classifiers.
Majority Voting

- The simplest ensemble method trains **different classifiers** on the **same training data** and uses **majority voting** to determine the class of an unseen data point.
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score

# load data
cars = pd.read_csv("/path-to/auto-mpg.data", sep="\s+", header=None)

# extract power and weight as data matrix X
X = cars.iloc[:, [3, 4]].values

# extract origin as target vector y
y = cars.iloc[:, 7].values

# split into training data (80%) and test data (20%)  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# fit logistic regression model on training data
lr = LogisticRegression()

# use kNN with k = 3
knn = KNeighborsClassifier(n_neighbors=3)

# learn decision tree
tree = DecisionTreeClassifier(criterion='entropy')

# voting classifier
vc = VotingClassifier(estimators=[('lr', lr), ('knn', knn), ('tree', tree)], voting='hard')
vc.fit(X_train, y_train)
vc_y_predicted = vc.predict(X_test)

print(accuracy_score(y_true=y_test, y_pred=vc_y_predicted))  # 0.759493670886
Bagging (bootstrap aggregation) trains different classifiers on samples of the training data and uses majority voting to determine the class of an unseen data point.
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import accuracy_score

# load data
cars = pd.read_csv("/path-to/auto-mpg.data",sep="\s+",header=None)

# extract power and weight as data matrix X
X = cars.iloc[:, [3,4]].values

# extract origin as target vector y
y = cars.iloc[:, 7].values

# split into training data (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# learn decision tree of maximal depth 2
tree = DecisionTreeClassifier(criterion='entropy', max_depth=2)

# bagging classifier based on 10 decision trees
bc = BaggingClassifier(base_estimator=tree, n_estimators=10)
bc.fit(X_train, y_train)
bc_y_predicted = bc.predict(X_test)

print(accuracy_score(y_true=y_test, y_pred=bc_y_predicted)) # 0.683544303797
Boosting

- Boosting trains a **sequence of classifiers**, so that later classifiers are trained on data points that have been misclassified by previous classifiers
  - **Classifier 1** is trained on a random sample of training data
  - **Classifier 2** is trained on a random sample of training data enriched by **50% of data points** misclassified by **Classifier 1**
  - **Classifier 3** is trained on a random sample of data points from the training data, for which **Classifier 1 and Classifier 2 disagree**, i.e., predict different classes

- Class of unseen data point is predict based on a **majority vote** of **Classifiers 1, 2, and 3**
Boosting

Training Data

Sample 1

Classifier 1

Prediction 1

Sample 2

Classifier 2

Prediction 2

Sample 3

Classifier 3

Prediction 3

Prediction

disagreement

misclassified
AdaBoost, as a popular variant of boosting, trains a sequence of classifiers; it increases the weight of data points that have been misclassified by the ensemble consisting of the already-trained classifiers, i.e., later classifiers learn to correct the mistakes of earlier ones.
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score

# load data
cars = pd.read_csv("/path-to/auto-mpg.data", sep="\s+", header=None)

# extract power and weight as data matrix X
X = cars.iloc[:, [3, 4]].values

# extract origin as target vector y
y = cars.iloc[:, 7].values

# split into training data (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# learn decision tree of maximal depth 2
tree = DecisionTreeClassifier(criterion='entropy', max_depth=2)

# bagging classifier based on 10 decision trees
bc = AdaBoostClassifier(base_estimator=tree, n_estimators=50)
bc.fit(X_train, y_train)
bc_y_predicted = bc.predict(X_test)

print(accuracy_score(y_true=y_test, y_pred=bc_y_predicted))  # 0.759493670886
Summary

- **Decision trees** as a classification method whose predictions are *easily interpretable* for humans

- **Information theory** provides a foundation for choosing decision criteria for nodes in decision trees

- **Majority voting, bagging, and boosting** as *ensemble methods* that aggregate the predictions of multiple classifiers to obtain a more robust classifier
References
