Admin

Assignment 1 due tomorrow (Friday)

Assignment 2 out soon: start ASAP! (due next Sunday)
  - Can (and are encouraged to) work in pairs

Slack

Office hours M-Th, 2:30-3:30pm, starting today (zoom link in sakai)

Representing examples

What is an example?

How is it represented?
Features

- Examples: red, round, leaf, 3oz, ...
- Features: red, round, leaf, 3oz, ...
- Features are the questions we can ask about the examples.

How our algorithms actually "view" the data.

Classification revisited

- Examples: red, round, leaf, 3oz, ...
- Label: apple
- Model/Classifier: learn

During learning/training/induction, learn a model of what distinguishes apples and bananas based on the features.

The model can then classify a new example based on the features.

Predict: Apple or banana?
Classification revisited

The model can then classify a new example based on the features.

Why?

Classification revisited

Learning is about generalizing from the training data

What does this assume about the training and test set?
Decision trees

Tree with internal nodes labeled by features
Branches are labeled by tests on that feature
Leaves labeled with classes

Leaves at 8 AM
Weather = Rainy
Accident = Yes
Stall = No

Decision trees

Tree with internal nodes labeled by features
Branches are labeled by tests on that feature
Leaves labeled with classes

Leaves at 10 AM
Weather = Rainy
Accident = No
Stall = No
**Decision trees**

Tree with internal nodes labeled by features

Branches are labeled by tests on that feature

Leaves labeled with classes

**To ride or not to ride, that is the question...**

<table>
<thead>
<tr>
<th>Terrain</th>
<th>Unicycle-type</th>
<th>Weather</th>
<th>Go-For-Ride?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Rainy</td>
<td>NO</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>Road</td>
<td>Mountain</td>
<td>Rainy</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Snowy</td>
<td>YES</td>
</tr>
</tbody>
</table>

**Recursive approach**

Base case: If all data belong to the same class, create a leaf node with that label

Otherwise:
- calculate the “score” for each feature if we used it to split the data
- pick the feature with the highest score, partition the data based on that data value and call recursively

**Partitioning the data**

<table>
<thead>
<tr>
<th>Terrain</th>
<th>Unicycle-type</th>
<th>Weather</th>
<th>Go-For-Ride?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Rainy</td>
<td>NO</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>Road</td>
<td>Mountain</td>
<td>Rainy</td>
<td>YES</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Snowy</td>
<td>YES</td>
</tr>
</tbody>
</table>
Partitioning the data

<table>
<thead>
<tr>
<th>Terrain</th>
<th>Unicycle-type</th>
<th>Weather</th>
<th>Go For Ride?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Rainy</td>
<td>NO</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Rainy</td>
<td>YES</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Snowy</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Snowy</td>
<td>YES</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Rainy</td>
<td>YES</td>
</tr>
<tr>
<td>Road</td>
<td>Mountain</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Snowy</td>
<td>YES</td>
</tr>
</tbody>
</table>

YES: 4  NO: 1
Partitioning the data

<table>
<thead>
<tr>
<th>Terrain</th>
<th>Unicycle-type</th>
<th>Weather</th>
<th>Can-Fun</th>
<th>Ride?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Rainy</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Sunny</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Sunny</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Road</td>
<td>Mountain</td>
<td>Rainy</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Road</td>
<td>Mountain</td>
<td>Snowy</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Snowy</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Road</td>
<td>Mountain</td>
<td>Snowy</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Sunny</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Snowy</td>
<td>YES</td>
<td></td>
</tr>
</tbody>
</table>

25

26

27

28
How could we make these into decision trees?

Training error: the average error over the training set

For classification, the most common “error” is the number of mistakes

Training error for each of these?
Training error vs. accuracy

**Terrain**
- Road: YES 4, NO 1
- Trail: YES 2, NO 3

**Unicycle**
- Normal: YES 4, NO 0
- Sunny: YES 2, NO 4

**Weather**
- Rainy: YES 2, NO 1
- Snowy: YES 2, NO 1

Training error: 3/10  
Training accuracy: 7/10

---

Recurse

**Terrain**
- Road: YES 4, NO 0
- Trail: YES 2, NO 4

**Unicycle**
- Normal: YES 4, NO 0
- Mountain: YES 2, NO 4

**Weather**
- Sunny: YES 4, NO 0
- Rainy: YES 2, NO 4

Training error: 3/10  
Training accuracy: 7/10

---

Recurse

**Terrain**
- Road: YES 4, NO 0
- Trail: YES 2, NO 4

**Unicycle**
- Normal: YES 4, NO 0
- Mountain: YES 2, NO 4

**Weather**
- Sunny: YES 4, NO 0
- Rainy: YES 2, NO 4

Training error: 3/10  
Training accuracy: 7/10

---

What should we do?
No need to examine other features since all examples have the same label.

Still two features left we can split on.
Recurse

**Unicycle**

Mountain

Normal

**Terrain**

Road

Trail

YES: 4

NO: 4

YES: 2

NO: 0

YES: 0

NO: 3

1/20/22

11

Which should we pick?

1/6

Recurse

**Unicycle**

Mountain

Normal

**Terrain**

Road

Trail

YES: 4

NO: 4

YES: 2

NO: 1

YES: 0

NO: 3

Recurse

**Unicycle**

Mountain

Normal

**Terrain**

Road

Trail

YES: 4

NO: 0

YES: 2

NO: 1

YES: 0

NO: 3

Which should we pick?

1/6

Recurse

**Unicycle**

Mountain

Normal

**Terrain**

Road

Trail

YES: 4

NO: 0

YES: 2

NO: 1

YES: 0

NO: 3

Recurse

**Unicycle**

Mountain

Normal

**Terrain**

Road

Trail

YES: 4

NO: 0

YES: 2

NO: 1

YES: 0

NO: 3

Which should we pick?

1/6

Recurse

**Unicycle**

Mountain

Normal

**Terrain**

Road

Trail

YES: 4

NO: 0

YES: 2

NO: 1

YES: 0

NO: 3

Recurse

**Unicycle**

Mountain

Normal

**Terrain**

Road

Trail

YES: 4

NO: 0

YES: 2

NO: 1

YES: 0

NO: 3

Which should we pick?
Problematic data

Recursive approach

Base case: If all data belong to the same class, create a leaf node with that label OR all the data has the same feature values

Do we always want to go all the way to the bottom?
What would the tree look like for...

<table>
<thead>
<tr>
<th>Terrain</th>
<th>Unicycle-type</th>
<th>Weather</th>
<th>Go-For-Ride?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Rainy</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>Road</td>
<td>Mountain</td>
<td>Snowy</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Sunny</td>
<td>NO</td>
</tr>
</tbody>
</table>

Is that what you would do?

An aside: how did we decide to pick the label for normal—road—rainy?
Overfitting occurs when we bias our model too much towards the training data. Our goal is to learn a general model that will work on the training data as well as other data (i.e., test data).
Overfitting

Even though the training error is decreasing, the testing error can go up!

Overfitting

How do we prevent overfitting?

Preventing overfitting

Base case:
- If all data belong to the same class, create a leaf node with that label
- OR all the data has the same feature values
- OR We’ve reached a particular depth in the tree
- ?

One idea: stop building the tree early

Preventing overfitting

Base case:
- If all data belong to the same class, create a leaf node with that label
- OR all the data has the same feature values
- OR We’ve reached a particular depth in the tree
- We only have a certain number/fraction of examples remaining
- We’ve reached a particular training error
- Use development data (more on this later)
- ...
Preventing overfitting: pruning

Pruning: after the tree is built, go back and “prune” the tree, i.e. remove some lower parts of the tree

Similar to stopping early, but done after the entire tree is built

Pruning criterion?
Handling non-binary attributes

What do we do with features that have multiple values? Real-values?

Features with multiple values

Real-valued features

Use any comparison test (>, <, ≤, ≥) to split the data into two parts

Select a range filter, i.e. min < value < max

Other splitting criterion

Otherwise:
- calculate the “score” for each feature if we used it to split the data
- pick the feature with the highest score, partition the data based on that data value and call recursively

We used training error for the score. Any other ideas?
Other splitting criterion

- Entropy: how much uncertainty there is in the distribution over labels after the split
- Gini: sum of the square of the label proportions after split
- Training error = misclassification error

Decision trees

Good? Bad?

Decision trees: the good

Very intuitive and easy to interpret

Fast to run and fairly easy to implement (Assignment 2 😋)

Historically, perform fairly well (especially with a few more tricks we’ll see later on)

No prior assumptions about the data

Decision trees: the bad

Be careful with features with lots of values if you’re not doing binary splits

<table>
<thead>
<tr>
<th>ID</th>
<th>Terrain</th>
<th>Unicycle Type</th>
<th>Weather</th>
<th>Go/No-Go?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Trail</td>
<td>Normal</td>
<td>Rainy</td>
<td>NO</td>
</tr>
<tr>
<td>2</td>
<td>Road</td>
<td>Normal</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>3</td>
<td>Trail</td>
<td>Mountain</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>4</td>
<td>Road</td>
<td>Mountain</td>
<td>Rainy</td>
<td>YES</td>
</tr>
<tr>
<td>5</td>
<td>Trail</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>6</td>
<td>Road</td>
<td>Normal</td>
<td>Rainy</td>
<td>YES</td>
</tr>
<tr>
<td>7</td>
<td>Road</td>
<td>Mountain</td>
<td>Snowy</td>
<td>YES</td>
</tr>
<tr>
<td>8</td>
<td>Trail</td>
<td>Normal</td>
<td>Sunny</td>
<td>NO</td>
</tr>
<tr>
<td>9</td>
<td>Road</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>10</td>
<td>Trail</td>
<td>Mountain</td>
<td>Snowy</td>
<td>YES</td>
</tr>
</tbody>
</table>

Which feature would be at the top here?
Decision trees: the bad

- Can be problematic (slow, bad performance) with large numbers of features
- Can't learn some very simple data sets (e.g. some types of linearly separable data)
- Pruning/tuning can be tricky to get right

Final DT algorithm

```
DT_train(data):

Base cases:
1. If all data belong to the same class, pick that label
2. If all the data have the same feature values, pick majority label
3. If we're out of features to examine, pick majority label
4. If the we don't have any data left, pick majority label of parent
5. If some other stopping criteria exists to avoid overfitting, pick majority label

 Otherwise (i.e. if none of the base cases apply):
- Calculate the "score" for each feature if we used it to split the data
- Pick the feature with the highest score, partition the data based on that data, e.g. data_left and data_right
- Recurse, i.e. DT_train(data_left) and DT_train(data_right)
- Make tree with feature as the splitting criterion with the decision trees returned from the recursive calls as the children
```