Assignment 1 due tomorrow (Friday)
Assignment 2 out soon: start ASAP! (due next Sunday)
- Can (and are STRONGLY encouraged to) work in pairs
- Slack

Features
- How our algorithms actually "view" the data
- Features are the questions we can ask about the examples

Admin
- Mentor hours this week:
  - Thursday (Today), 7-9pm (Edmunds upstairs)
- Mentor hours starting next week:
  - Friday, 7-9pm
  - Sunday, 7-9pm
- Lecture notes posted (webpage)
- Keep up with the reading
- Videos before class

Admin
- Assignment 1 due tomorrow (Friday)
- Assignment 2 out soon: start ASAP! (due next Sunday)
- Can (and are STRONGLY encouraged to) work in pairs
- Slack
A sample data set

<table>
<thead>
<tr>
<th>Time</th>
<th>Weather</th>
<th>Stall</th>
<th>Accident</th>
<th>Commute</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 AM</td>
<td>Sunny</td>
<td>Yes</td>
<td>No</td>
<td>Long</td>
</tr>
<tr>
<td>8 AM</td>
<td>Cloudy</td>
<td>Yes</td>
<td>No</td>
<td>Long</td>
</tr>
<tr>
<td>10 AM</td>
<td>Sunny</td>
<td>Yes</td>
<td>Yes</td>
<td>Long</td>
</tr>
<tr>
<td>9 AM</td>
<td>Rainy</td>
<td>Yes</td>
<td>No</td>
<td>Long</td>
</tr>
<tr>
<td>10 AM</td>
<td>Sunny</td>
<td>Yes</td>
<td>Yes</td>
<td>Long</td>
</tr>
<tr>
<td>9 AM</td>
<td>Sunny</td>
<td>Yes</td>
<td>No</td>
<td>Long</td>
</tr>
<tr>
<td>10 AM</td>
<td>Cloudy</td>
<td>Yes</td>
<td>No</td>
<td>Long</td>
</tr>
<tr>
<td>9 AM</td>
<td>Sunny</td>
<td>Yes</td>
<td>No</td>
<td>Long</td>
</tr>
<tr>
<td>10 AM</td>
<td>Cloudy</td>
<td>Yes</td>
<td>Yes</td>
<td>Long</td>
</tr>
<tr>
<td>8 AM</td>
<td>Cloudy</td>
<td>Yes</td>
<td>No</td>
<td>Long</td>
</tr>
<tr>
<td>9 AM</td>
<td>Rainy</td>
<td>No</td>
<td>No</td>
<td>Short</td>
</tr>
<tr>
<td>10 AM</td>
<td>Cloudy</td>
<td>No</td>
<td>No</td>
<td>Short</td>
</tr>
<tr>
<td>10 AM</td>
<td>Sunny</td>
<td>Yes</td>
<td>No</td>
<td>Long</td>
</tr>
<tr>
<td>10 AM</td>
<td>Rainy</td>
<td>No</td>
<td>No</td>
<td>Short</td>
</tr>
</tbody>
</table>

Can you describe a “model” that could be used to make decisions in general?

**Decision trees**

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

**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</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>Rainy</td>
<td>YES</td>
</tr>
<tr>
<td>Road</td>
<td>Mountain</td>
<td>Snowy</td>
<td>YES</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>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>Mountain</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>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Rainy</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Snowy</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Sunny</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>
Partitioning the data

<table>
<thead>
<tr>
<th>Terrain</th>
<th>Road</th>
<th>Trail</th>
<th>Unicycle</th>
<th>Normal</th>
<th>Rainy</th>
<th>Sunny</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES: 4</td>
<td>YES: 2</td>
<td>YES: 4</td>
<td>YES: 2</td>
<td>YES: 2</td>
<td>YES: 2</td>
<td>YES: 2</td>
</tr>
<tr>
<td>NO: 0</td>
<td>NO: 3</td>
<td>NO: 0</td>
<td>NO: 4</td>
<td>NO: 1</td>
<td>NO: 2</td>
<td>NO: 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Terrain</th>
<th>Road</th>
<th>Trail</th>
<th>Unicycle</th>
<th>Normal</th>
<th>Rainy</th>
<th>Sunny</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES: 4</td>
<td>YES: 2</td>
<td>YES: 4</td>
<td>YES: 2</td>
<td>YES: 2</td>
<td>YES: 2</td>
<td>YES: 2</td>
</tr>
<tr>
<td>NO: 0</td>
<td>NO: 3</td>
<td>NO: 0</td>
<td>NO: 4</td>
<td>NO: 1</td>
<td>NO: 2</td>
<td>NO: 1</td>
</tr>
</tbody>
</table>

calculate the “score” for each feature
if we used it to split the data

What score should we use?
If we just stopped here, which tree would be best?
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?
Decision trees

Training error vs. accuracy

Training error: the average error over the training set

Training accuracy: the average proportion correct over the training set

Recurse

Recurse
Recurse

Unicycle

Mountain

Normal

YES: 4
NO: 0

Terrain

Unicycle -

Weather

Go -

Ride?

Trail

Mountain: Sunny

YES

Road

Mountain: Rainy

YES

Road

Mountain: Sunny

YES

Trail

Mountain: Sunny

YES

What should we do?

29

No need to examine other features since all examples have the same label.

30

Still two features left we can split on

31

32
Which should we pick?
Recurse

Unicycle

Mountain

Normal

YES: 4
NO: 0

Road

Trail

YES: 2
NO: 1

YES: 0

Terrain

Unicycle - type

Weather

Go - For Ride?

Road

Normal

Sunny

YES

Road

Normal

Rainy

YES

Road

Normal

Snowy

NO

Terrain

Road

Trail

YES: 2
NO: 1

YES: 0

NO: 3

Weather

Rainy

Sunny

YES

NO

YES: 1

NO: 1

YES: 1

NO: 0

YES: 0

NO: 1

Training error?

Are we always guaranteed to get a training error of 0?

Problematic data

Terrain

Unicycle

Mountain

Normal

YES: 4
NO: 0

Road

Trail

YES: 2
NO: 1

YES: 0

Weather

Rainy

Sunny

YES

NO

YES: 1

NO: 1

YES: 1

NO: 0

YES: 0

NO: 1

When can this happen?
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></th>
<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>
<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>Sunny</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>Trail</td>
<td>Normal</td>
<td>Rainy</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Snowy</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Sunny</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Sunny</td>
<td>NO</td>
<td></td>
</tr>
</tbody>
</table>

Maybe...

<table>
<thead>
<tr>
<th></th>
<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>
<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>Sunny</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>Trail</td>
<td>Normal</td>
<td>Rainy</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Snowy</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Sunny</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Sunny</td>
<td>NO</td>
<td></td>
</tr>
</tbody>
</table>
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

Test set error!

Machine learning is about predicting the future based on the past.

— Hal Daume III

Even though the training error is decreasing, the testing error can go up!
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
- OR We’ve reached a particular number/fraction of examples remaining
- OR We’ve reached a particular training error
- Use development data (more on this later)
...

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
Preventing overfitting: pruning

Build the full tree

Prune back leaves that are too specific

Pruning criterion?

Handling non-binary attributes

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

- Treat as an n-ary split
- Treat as multiple binary splits

Real-valued features

- Use any comparison test ($>$, $<$, $\leq$, $\geq$) 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?
Decision trees

**Good? Bad?**

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>Height</th>
<th>Weather</th>
<th>Ride</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>
</tbody>
</table>

Which feature would be at the top here?

---

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

```plaintext
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 (if tie, parent majority)
3. If we're out of features to examine, pick majority label (if tie, parent majority)
4. If 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.
```