Assignment 2 experiments

- How good was the decision tree?
- How deep did it need to be?
- Overfitting?
- Training data size?

Features

<table>
<thead>
<tr>
<th>Terrain</th>
<th>Unicycle-type</th>
<th>Weather</th>
<th>Go-Far-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>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>

Where do they come from?
UCI Machine Learning Repository


Provided features

Predicting the age of abalone from physical measurements

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Measurement Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sex</td>
<td>nominal</td>
<td>--</td>
<td>M, F, and I (infant)</td>
</tr>
<tr>
<td>Length</td>
<td>continuous</td>
<td>mm</td>
<td>Longest shell measurement</td>
</tr>
<tr>
<td>Diameter</td>
<td>continuous</td>
<td>mm</td>
<td>Perpendicular to length</td>
</tr>
<tr>
<td>Height</td>
<td>continuous</td>
<td>mm</td>
<td>With meat in shell</td>
</tr>
<tr>
<td>Whole weight</td>
<td>continuous</td>
<td>grams</td>
<td>Whole abalone</td>
</tr>
<tr>
<td>Shucked weight</td>
<td>continuous</td>
<td>grams</td>
<td>Weight of meat</td>
</tr>
<tr>
<td>Viscera weight</td>
<td>continuous</td>
<td>grams</td>
<td>Gut weight (after bleeding)</td>
</tr>
<tr>
<td>Shell weight</td>
<td>continuous</td>
<td>grams</td>
<td>After being dried</td>
</tr>
<tr>
<td>Rings</td>
<td>integer</td>
<td>--</td>
<td>+1.5 gives the age in years</td>
</tr>
</tbody>
</table>

Provided features

Predicting breast cancer recurrence

1. Class: no-recurrence-events, recurrence-events
3. menopause: premeno, perimen, postmeno.
5. inv-nodes: 0-2, 3-5, 6-8, 9-11, 12-14, 15-17, 18-20, 21-23, 24-26, 27-29, 30-32, 33-35, 36-39.
6. node-caps: yes, no.
7. deg-marg: 1, 2, 3.
8. breast: left, right.
9. breast-quad: left-up, left-low, right-up, right-low, central.
10. irradiated: yes, no.

Provided features

In many physical domains (e.g. biology, medicine, chemistry, engineering, etc.)
- the data has been collected and the relevant features have been identified
- we cannot collect more features from the examples (at least "core" features)

In these domains, we can often just use the provided features
Raw data vs. features

In other domains, we are provided with the raw data, but must extract/identify features.

For example:
- image data
- text data
- audio data
- log data
- ...

How is an image represented?

• images are made up of pixels
• for a color image, each pixel corresponds to an RGB value (i.e. three numbers)

Image features

for each pixel:
R[0-255]
G[0-255]
B[0-255]

Do we retain all the information in the original document?
Image features

- For each pixel: $R[0-255]$, $G[0-255]$, $B[0-255]$

Other features for images?

Lots of image features

- Use “patches” rather than pixels (sort of like “bigrams” for text)
- Different color representations (i.e. L*A*B*)
- Texture features, i.e. responses to filters
- Shape features
- …

Obtaining features

Very often requires some domain knowledge

As ML algorithm developers, we often have to trust the “experts” to identify and extract reasonable features

That said, it can be helpful to understand where the features are coming from

Current learning model

Training data (labeled examples)

Learn

Model/Classifier
Pre-process training data

Outlier detection

Outlier detection

Outlier detection

Outlier detection
Outlier detection

An example that is inconsistent with the other examples
- extreme feature values in one or more dimensions
- examples with the same feature values but different labels

Fix

Removing conflicting examples

Identify examples that have the same features, but differing values
- For some learning algorithms, these examples can cause issues (for example, not converging)
- In general, unsatisfying from a learning perspective

Can be a bit expensive computationally (examining all pairs), though faster approaches are available

Removing extreme outliers

Throw out examples that have extreme values in one dimension

Throw out examples that are very far away from any other example

Train a probabilistic model on the data and throw out “very unlikely” examples

This is an entire field of study by itself! Often called outlier or anomaly detection.
Quick statistics recap

What are the mean, standard deviation, and variance of data?

mean: average value, often written as \( \mu \)

variance: a measure of how much variation there is in the data. Calculated as:

\[
\sigma^2 = \frac{\sum (x_i - \mu)^2}{n-1}
\]

standard deviation: square root of the variance (written as \( \sigma \))

How can these help us with outliers?

Outlier detection

If we know the data is distributed normally (i.e. via a normal/gaussian distribution)

Outliers in a single dimension

Examples in a single dimension that have values greater than \(|k\sigma|\) can be discarded (for \(k >> 3\))

Even if the data isn’t actually distributed normally, this is still often reasonable
Outliers for machine learning

Some good practices:
- Throw out conflicting examples
- Throw out any examples with obviously extreme feature values (i.e., many, many standard deviations away)
- Check for erroneous feature values (e.g., negative values for a feature that can only be positive)
- Let the learning algorithm/other pre-processing handle the rest

So far...

1. Throw out outlier examples
2. Which features to use

Feature pruning/selection

Good features provide us with information that helps us distinguish between labels. However, not all features are good.

Feature pruning is the process of removing “bad” features

Feature selection is the process of selecting “good” features

What makes a bad feature and why would we have them in our data?

Bad features

Each of you are going to generate a feature for our data set: pick 5 random binary numbers

$f_1 f_2 ... \quad \text{label}$

I’ve already labeled these examples and I have two features
If we have a “random” feature, i.e. a feature with random binary values, what is the probability that our feature perfectly predicts the label?

<table>
<thead>
<tr>
<th>label</th>
<th>f_i</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.5</td>
</tr>
</tbody>
</table>

0.5^5 = 0.03125 = 1/32

Is that the only way to get perfect prediction?

Total = 1/32 + 1/32 = 1/16

Why is this a problem?

Although these features perfectly correlate/predict the training data, they will generally NOT have any predictive power on the test set!

Is perfect correlation the only thing we need to worry about for random features?
Bad features

<table>
<thead>
<tr>
<th>label</th>
<th>f_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Any correlation (particularly any strong correlation) can affect performance!

Noisy features

Adding features can give us more information, but not always

Determining if a feature is useful can be challenging

<table>
<thead>
<tr>
<th>Terrain</th>
<th>MTurkType</th>
<th>Weather</th>
<th>gender</th>
<th>Age</th>
<th>ML grade</th>
<th>Go For Ride?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Rainy</td>
<td>Heavy</td>
<td>D</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Sunny</td>
<td>Light</td>
<td>C</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Road</td>
<td>Mountain</td>
<td>Sunny</td>
<td>Light</td>
<td>B</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Road</td>
<td>Mountain</td>
<td>Sunny</td>
<td>Heavy</td>
<td>A</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Snowy</td>
<td>Light</td>
<td>D+</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Rainy</td>
<td>Heavy</td>
<td>B</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Snowy</td>
<td>Heavy</td>
<td>C+</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Snowy</td>
<td>Light</td>
<td>A</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Snowy</td>
<td>Light</td>
<td>B+</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Rainy</td>
<td>Light</td>
<td>C</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Noisy features

These can be particularly problematic in problem areas where we automatically generate features

Ideas for removing noisy/random features?
Removing noisy features

The expensive way:
- Split training data into train/dev
- Train a model on all features
- for each feature f:
  - Train a model on all features minus f
  - Compare performance of all vs. all-f on dev set
- Remove all features where decrease in performance between all and all-f is less than some constant

Feature ablation study  Issues/concerns?

Removing noisy features

Binary features:
remove “rare” features, i.e. features that only occur (or don’t occur) a very small number of times

Real-valued features:
remove features that have low variance

In both cases, can either use thresholds, throw away lowest x%, use development data, etc.

Why?

Some rules of thumb for the number of features

Be very careful in domains where:
- the number of features > number of examples
- the number of features = number of examples
- the features are generated automatically
- there is a chance of “random” features

In most of these cases, features should be removed based on some domain knowledge (i.e. problem-specific knowledge)

So far…

1. Throw out outlier examples
2. Remove noisy features
3. Pick “good” features
Feature selection

Let’s look at the problem from the other direction, that is, selecting good features.

What are good features?

How can we pick/select them?

Good features

A good feature correlates well with the label

<table>
<thead>
<tr>
<th>label</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

How can we identify this?

- training error (like for DT)
- correlation model
- statistical test
- probabilistic test
- ...

Training error feature selection

- for each feature f:
  - calculate the training error if only feature f were used to pick the label
  - rank each feature by this value
  - pick top k, top x%, etc.
  - can use a development set to help pick k or x

So far...

1. Throw out outlier examples
2. Remove noisy features
3. Pick “good” features
Would our three classifiers (DT, k-NN and perceptron) learn the same models on these two data sets?

Decision trees don’t care about scale, so they’d learn the same tree.

k-NN: NO! The distances are biased based on feature magnitude.

\[ D(a, b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2} \]
Feature normalization

<table>
<thead>
<tr>
<th>Length</th>
<th>Weight</th>
<th>Label</th>
<th>D(a, b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2}</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>Apple</td>
<td>(D = \sqrt{(7 - 4)^2 + (3 - 4)^2} = \sqrt{18})</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>Apple</td>
<td>(D = \sqrt{(5 - 4)^2 + (8 - 4)^2} = \sqrt{17})</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>Banana</td>
<td>(D = \sqrt{(70 - 40)^2 + (5 - 4)^2} = \sqrt{901})</td>
</tr>
<tr>
<td>40</td>
<td>4</td>
<td>Apple</td>
<td>(D = \sqrt{(70 - 40)^2 + (5 - 4)^2} = \sqrt{901})</td>
</tr>
<tr>
<td>70</td>
<td>5</td>
<td>Apple</td>
<td>(D = \sqrt{(70 - 50)^2 + (8 - 4)^2} = \sqrt{416})</td>
</tr>
</tbody>
</table>

Geometric view of perceptron update

for each \(w_i\):
\(w_i = w_i + f \cdot \text{label}\)

Geometrically, the perceptron update rule is equivalent to “adding” the weight vector and the feature vector.

D(a, b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2}
Geometric view of perceptron update

If the features dimensions differ in scale, it can bias the update

- different separating hyperplanes
- the larger dimension becomes much more important

How do we fix this?

Feature normalization

<table>
<thead>
<tr>
<th>Length</th>
<th>Weight</th>
<th>Color</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>Apple</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>Apple</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>1</td>
<td>Banana</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>0</td>
<td>Apple</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>1</td>
<td>Banana</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>1</td>
<td>Banana</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1</td>
<td>Apple</td>
</tr>
</tbody>
</table>

Modify all values for a given feature
Normalize each feature

For each feature (over all examples):

Center: adjust the values so that the mean of that feature is 0. How do we do this?

Rescale/adjust feature values to avoid magnitude bias. Ideas?

So far...

1. Throw out outlier examples
2. Remove noisy features
3. Pick "good" features
4. Normalize feature values
   1. center data
   2. scale data (either variance or absolute)
Example normalization

<table>
<thead>
<tr>
<th>Length</th>
<th>Weight</th>
<th>Color</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>Apple</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
<td>Apple</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>1</td>
<td>Banana</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>0</td>
<td>Apple</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>1</td>
<td>Banana</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>1</td>
<td>Banana</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1</td>
<td>Apple</td>
</tr>
</tbody>
</table>

Any problem with this? Solutions?

Example length normalization

Make all examples roughly the same scale, e.g. make all have length = 1

What is the length of this example/vector?

\[ \|x\| = \sqrt{x_1^2 + x_2^2} \]

Example length normalization

Make all examples roughly the same scale, e.g. make all have length = 1

What is the length of this example/vector?

\[ \|x\| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2} \]
Example length normalization

Make all examples have length = 1

Divide each feature value by \( ||x|| \)

- Prevents a single example from being too impactful
- Equivalent to projecting each example onto a unit sphere

\[
\text{length}(x) = ||x|| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}
\]

So far...

1. Throw out outlier examples
2. Remove noisy features
3. Pick “good” features
4. Normalize feature values
   1. center data
   2. scale data (either variance or absolute)
5. Normalize example length
6. Finally, train your model!