An aside: text classification

<table>
<thead>
<tr>
<th>Raw data</th>
<th>Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Chardonnay</td>
</tr>
<tr>
<td></td>
<td>Pinot Grigio</td>
</tr>
<tr>
<td></td>
<td>Zinfandel</td>
</tr>
</tbody>
</table>

Text: raw data

<table>
<thead>
<tr>
<th>Raw data</th>
<th>Labels</th>
<th>Features?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Chardonnay</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pinot Grigio</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Zinfandel</td>
<td></td>
</tr>
</tbody>
</table>
**Feature examples**

<table>
<thead>
<tr>
<th>Raw data</th>
<th>labels</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chardonnay</td>
<td>Clinton said pinot repeatedly last week on tv, “pinot, pinot, pinot”.</td>
<td></td>
</tr>
<tr>
<td>Pinot Grigio</td>
<td>(1, 1, 0, 1, 0, 0, 0, 0, ...)</td>
<td></td>
</tr>
<tr>
<td>Zinfandel</td>
<td>Occurrence of words</td>
<td></td>
</tr>
</tbody>
</table>

**Feature examples**

<table>
<thead>
<tr>
<th>Raw data</th>
<th>labels</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chardonnay</td>
<td>Clinton said pinot repeatedly last week on tv, “pinot, pinot, pinot”.</td>
<td></td>
</tr>
<tr>
<td>Pinot Grigio</td>
<td>(4, 1, 1, 0, 0, 1, 0, 0, ...)</td>
<td></td>
</tr>
<tr>
<td>Zinfandel</td>
<td>Frequency of word occurrences</td>
<td></td>
</tr>
</tbody>
</table>

This is the representation we’re using for assignment 5

**Decision trees for text**

Each internal node represents whether or not the text has a particular word

- **wheat**
  - bushel
  - export
  - commodity
  - farm
  - wheat
  - Not wheat

- **wheat**
  - export
  - commodity
  - farm
  - wheat
  - Not wheat

- **wheat**
  - export
  - commodity
  - agriculture
  - wheat
  - Not wheat
The US views technology as a commodity that it can export by the bushel.

Printing out decision trees

A high-bias assumption is linear separability:
- in 2 dimensions, can separate classes by a line
- in higher dimensions, need hyperplanes

A linear model is a model that assumes the data is linearly separable.
Linear models

A linear model in \(n\)-dimensional space (i.e. \(n\) features) is defined by \(n+1\) weights:

In two dimensions, a line:
\[ 0 = w_1 f_1 + w_2 f_2 + b \] (where \(b = -a\))

In three dimensions, a plane:
\[ 0 = w_1 f_1 + w_2 f_2 + w_3 f_3 + b \]

In \(m\)-dimensions, a hyperplane
\[ 0 = b + \sum_{j=1}^{m} w_j f_j \]

Perceptron learning algorithm

repeat until convergence (or for some # of iterations):
for each training example \((f_1, f_2, \ldots, f_n, \text{label})\):
\[
\text{prediction} = b + \sum_{j=1}^{n} w_j f_j
\]

if prediction * label \(\leq 0\): // they don’t agree
for each \(w_j\):
\[
w_j = w_j + f_j \times \text{label}
\]
\[
b = b + \text{label}
\]

Which line will it find?

Only guaranteed to find some line that separates the data
Linear models

Perceptron algorithm is one example of a linear classifier.

Many, many other algorithms learn a line (i.e. a setting of a linear combination of weights).

Goals:
- Explore a number of linear training algorithms
- Understand why these algorithms work

Perceptron learning algorithm

repeat until convergence (or for some # of iterations);
for each training example \((f_1, f_2, \ldots, f_m, \text{label})\):

\[
\text{prediction} = b + \sum_{j=1}^{m} w_j f_j
\]

if \(\text{prediction} \times \text{label} \leq 0\) // they don’t agree
for each \(w_j\):

\[
w_i = w_j + f_i \times \text{label}
\]

\[
b = b + \text{label}
\]

A closer look at why we got it wrong

\[
w_1 \quad w_2
\]

\[
0 \times f_1 + 1 \times f_2 =
\]

\[
0 \times -1 + 1 \times -1 = -1
\]

We’d like this value to be positive since it’s a positive value.


didn’t contribute, but could have
\[
\text{decrease}
\]

\[
0 \to -1
\]

contributed in the wrong direction
\[
\text{decrease}
\]

\[
1 \to 0
\]

Intuitively these make sense:
- Why change by \(1\)?
- Any other way of doing it?

Model-based machine learning

1. pick a model
   - e.g. a hyperplane, a decision tree,…
   - A model is defined by a collection of parameters

What are the parameters for DTR? Perceptron?
Model-based machine learning

1. pick a model
   - e.g. a hyperplane, a decision tree,…
   - A model is defined by a collection of parameters

   DT: the structure of the tree, which features each node splits on, the predictions at the leaves
   perceptron: the weights and the b value

2. pick a criterion to optimize (aka objective function)
   - e.g. training error

3. develop a learning algorithm
   - the algorithm should try and minimize the criteria
   - sometimes in a heuristic way (i.e. non-optimally)
   - sometimes exactly

Linear models in general

$0 = b + \sum_{j=1}^{m} w_j f_j$

These are the parameters we want to learn

1. pick a model

   e.g. a hyperplane, a decision tree,…

   A model is defined by a collection of parameters

2. pick a criterion to optimize (aka objective function)
Some notation: indicator function

\[ f(x) = \begin{cases} 
1 & \text{if } x = \text{True} \\
0 & \text{if } x = \text{False} 
\end{cases} \]

Convenient notation for turning T/F answers into numbers/counts:

\[ \text{beers\_to\_bring\_for\_class} = \sum_{\text{age} \geq 21} 1 \]

Some notation: dot-product

Sometimes it is convenient to use vector notation

We represent an example \( f_1, f_2, \ldots, f_n \) as a single vector, \( x \)
- \( i \) subscript will indicate feature indexing, i.e., \( x_i \)
- \( k \) subscript will indicate examples indexing over a dataset, i.e., \( x_k \) or sometimes \( x_{ij} \)

Similarly, we can represent the weight vector \( w_1, w_2, \ldots, w_m \) as a single vector, \( w \)

The dot-product between two vectors \( a \) and \( b \) is defined as:

\[ a \cdot b = \sum_{j=1}^{n} a_j b_j \]

Linear models

1. pick a model

\[ 0 = b + \sum_{j=1}^{n} w_j x_j \]

These are the parameters we want to learn

2. pick a criterion to optimize (aka objective function)

\[ \sum_{i=1}^{n} [y_i (w \cdot x_i + b) \leq 0] \]

What does this equation say?

0/1 loss function

\[ \sum_{i=1}^{n} [y_i (w \cdot x_i + b) \leq 0] \]

- total number of mistakes, aka 0/1 loss
- distance from hyperplane
- sign is prediction
- whether or not the prediction and label agree, true if they don’t
10/1/19

Model-based machine learning

1. pick a model
   \[ 0 = b + \sum_{j=1}^{m} w_j f_j \]
2. pick a criteria to optimize (aka objective function)
   \[ \sum_{i=1}^{n} I[y_i (w \cdot x_i + b) \leq 0] \]
3. develop a learning algorithm
   \[ \text{argmin}_{w,b} \sum_{i=1}^{n} I[y_i (w \cdot x_i + b) \leq 0] \text{ Find w and b that minimize the 0/1 loss (i.e. training error)} \]

Minimizing 0/1 loss

Find w and b that minimize the 0/1 loss

How do we do this?
How do we minimize a function?
Why is it hard for this function?

Minimizing 0/1 in one dimension

Each time we change w such that the example is right/wrong the loss will increase/decrease

Minimizing 0/1 over all w

Each new feature we add (i.e. weights) adds another dimension to this space!
Minimizing 0/1 loss

Find \( w \) and \( b \) that minimize the 0/1 loss

\[
\arg\min_{w,b} \sum_{i=1}^{n} \left( y_i (w \cdot x_i + b) \leq 0 \right)
\]

This turns out to be hard (in fact, NP-HARD)

Challenge:
- small changes in any \( w \) can have large changes in the loss (the change isn’t continuous)
- there can be many, many local minima
- at any given point, we don’t have much information to direct us towards any minima

More manageable loss functions

What property/properties do we want from our loss function?

More manageable loss functions

Convex functions

Convex functions look something like:

- Ideally, continuous (i.e., differentiable) so we get an indication of direction of minimization
- Only one minima

One definition: The line segment between any two points on the function is above the function
Surrogate loss functions

For many applications, we really would like to minimize the 0/1 loss.

A surrogate loss function is a loss function that provides an upper bound on the actual loss function (in this case, 0/1).

We’d like to identify convex surrogate loss functions to make them easier to minimize.

Key to a loss function: how it scores the difference between the actual label \( y \) and the predicted label \( y' \).

### Surrogate loss functions

- **0/1 loss:** \( l(y, y') = [yy' \leq 0] \)
- **Hinge:** \( l(y, y') = \max(0,1 - yy') \)
- **Exponential:** \( l(y, y') = \exp(-yy') \)
- **Squared loss:** \( l(y, y') = (y - y')^2 \)

Why do these work? What do they penalize?
Model-based machine learning

1. pick a model
\[ 0 = b + \sum_{j=1}^{m} w_j f_j \]
2. pick a criteria to optimize (aka objective function)
\[ \sum_{i=1}^{n} \exp(-y_i (w \cdot x_i + b)) \]
   use a convex surrogate loss function
3. develop a learning algorithm
\[ \arg\min_{w} \sum_{i=1}^{n} \exp(-y_i (w \cdot x_i + b)) \]
   Find w and b that minimize the surrogate loss

Finding the minimum
You're blindfolded, but you can see out of the bottom of the blindfold to the ground right by your feet. I drop you off somewhere and tell you that you're in a convex shaped valley and escape is at the bottom/minimum. How do you get out?

Finding the minimum
How do we do this for a function?

One approach: gradient descent
Partial derivatives give us the slope (i.e. direction to move) in that dimension
One approach: gradient descent

Partial derivatives give us the slope (i.e. direction to move) in that dimension

Approach:
- pick a starting point \( (w) \)
- repeat:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)

Gradient descent

- pick a starting point \( (w) \)
- repeat until loss doesn't decrease in any dimension:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)

\[
w_j = w_j - \eta \frac{d}{dw_j} \text{loss}(w)
\]

Why negative?
Gradient descent

- pick a starting point \( (w) \)
- repeat until loss doesn’t decrease in any dimension:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)

\[
W_j = W_j - \eta \frac{d}{dW_j} \text{loss}(W)
\]

learning rate (how much we want to move in the error direction, often this will change over time)

Some math

\[
\frac{d}{dW_j} \text{loss} = \frac{d}{dW_j} \sum_{i=1}^{N} \exp(-y_i(w \cdot x_i + b))
\]

\[
= \sum_{i=1}^{N} \exp(-y_i(w \cdot x_i + b)) \frac{d}{dW_j} - y_i(w \cdot x_i + b)
\]

\[
= \sum_{i=1}^{N} -y_i x_{ij} \exp(-y_i(w \cdot x_i + b))
\]
Gradient descent

- pick a starting point ($w$)
- repeat until loss doesn’t decrease in any dimension:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)

$$w_j = w_j + \eta \sum_{i=1}^{n} y_i x_i^j \exp(-y_i (w \cdot x_i + b))$$

What is this doing?

Exponential update rule

$$w_j = w_j + \eta \sum_{i=1}^{n} y_i x_i^j \exp(-y_i (w \cdot x_i + b))$$

for each example $x_i$:

$$w_j = w_j + \eta y_i x_i^j \exp(-y_i (w \cdot x_i + b))$$

Does this look familiar?

Perceptron learning algorithm!

repeat until convergence (or for some $I$ of iterations):

for each training example ($f_1, f_2, ..., f_m$, label):

prediction = $b + \sum_{i=1}^{m} w_i f_i$

if prediction * label $\leq 0$: // they don’t agree

for each $w_j$

$w_j = w_j + \eta f_j \text{label}$

$b = b + \text{label}$

$$w_j = w_j + \eta y_i x_i^j \exp(-y_i (w \cdot x_i + b))$$

or

$$w_j = w_j + x_i^j y_i c \quad \text{where } c = \eta \exp(-y_i (w \cdot x_i + b))$$

The constant

$$c = \eta \exp(-y_i (w \cdot x_i + b))$$

learning rate  label  prediction

When is this large/small?
The constant $c = \eta \exp(-y_i(w \cdot x_i + b))$

If they’re the same sign, as the predicted gets larger there update gets smaller.

If they’re different, the more different they are, the bigger the update.

Perceptron learning algorithm!

repeat until convergence (or for some # of iterations): for each training example $(f_1, f_2, ..., f_m, \text{label})$

\[
\text{prediction} = b + \sum_{i=1}^{m} w_i f_i
\]

if prediction * label < 0 // they don’t agree

for each $w_i$

\[
w_i = w_i + f_i \times \text{label}
\]

$b = b + \text{label}$

\[
w_i = w_i + \eta y_i x_i \exp(-y_i(w \cdot x + b))
\]

or

\[
w_i = w_i + x_i y_i c
\]

where $c = \eta \exp(-y_i(w \cdot x + b))$

One concern

argmin_{w,b} \sum_{i=1}^{m} \exp(-y_i(w \cdot x_i + b))

We’re calculating this on the training set

We still need to be careful about overfitting!

The min $w,b$ on the training set is generally NOT the min for the test set.

How did we deal with this for the perceptron algorithm?

Summary

Model-based machine learning:
- define a model, objective function (i.e. loss function), minimization algorithm

Gradient descent minimization algorithm
- require that our loss function is convex
- make small updates towards lower losses

Perceptron learning algorithm:
- gradient descent
- exponential loss function (modulo a learning rate)