Natural Language Processing

CSCI 5832—Lecture 8

Jim Martin

NLP@CU

Today

- Pushing Quiz 1 back a week to Week 6. Same material.
- Wrap up evaluation
- Logistic Regression and Text Classification
  - Chapter 5

How do we know if one system is better than another?

Given

- Classifier A and B
- Metric M: M(A,x) is the performance of *A* on testset *x*
- δ(x): the performance difference between A, B on x:
  - $\delta(x) = M(A,x) - M(B,x)$

- We want to know if $\delta(x)>0$, meaning A is better than B
- δ(x) is called the **effect size**
- Suppose we look and see that δ(x) is positive. Are we done?
- No! This might be just an accident of this one test set, or circumstance of the experiment.

Statistical Hypothesis Testing

- Consider two hypotheses:
  - Null hypothesis: A isn't better than B
  - A is better than B

$$H_0 : \delta(x) \leq 0$$
$$H_1 : \delta(x) > 0$$

- To settle on H_1, we'll try to rule out H_0
  - That is, show that H_0 is very unlikely given the results
- We create a random variable X ranging over test sets
- And ask, if H_0 is true, how likely is it that among these test sets we would see the results we did see?
  - Formalized as the p-value

Statistical Hypothesis Testing

$$P(\delta(X) \geq \delta(x) | H_0 \text{ is true})$$

- The p-value is the probability that we would see δ assuming H_0 (*A* is not better than *B*).
  - If H_0 is true but δ is large, that is surprising! Very low probability!
- A very small p-value means that the difference we observed is very unlikely under the <u>null hypothesis</u>, and we can reject the null hypothesis
  - "Very small" is often .05 or .01
- A result("*A* is better than *B*") is **statistically significant** if the δ we saw has a probability that is below that threshold and we therefore reject this null hypothesis.

Statistical Hypothesis Testing

- How do we compute this probability?
- In NLP, we tend to use non-parametric tests based on sampling: artificially creating many versions of the setup.
- For example, suppose we had created zillions of testsets x'.
  - Now we measure the value of δ(x') on each test set
  - That gives us a distribution
  - Now set a threshold (say .01).
  - So if we see that in 99% of the test sets $\delta(x) > \delta(x')$
    - We conclude that our original test set delta was a real delta and not an artifact.

Statistical Hypothesis Testing

- Paired tests are a common approach used in NLP
  - Compare two sets of observations in which each observation in one set can be paired with an observation in another.
  - For example, when looking at systems A and B **on the same test set**, we can compare item for item the performance of system *A* and *B*

Bootstrap Test

- Choose a single metric (accuracy, precision, recall, F1, etc) for use in system evaluation
- Bootstrap means to repeatedly draw large numbers of smaller samples with replacement (called **bootstrap samples**) from an original sample.
  - Generate a large number of smaller tests from a single gold-standard test set.

Bootstrap Example

Consider a small text classification example with a test set x of 10 documents, using accuracy as metric.

Suppose these are the results of systems A and B on x, with 4 outcomes (A & B both right, A & B both wrong, A right/B wrong, A wrong/B right):

	1	2	3	4	5	6	7	8	9	10	A%	B%	$\delta()$
x	AB	$A\cancel{B}$	AB	$\cancel{A}B$	$A\cancel{B}$	$\cancel{A}B$	$A\cancel{B}$	AB	$\cancel{A}\cancel{B}$	$A\cancel{B}$	.70	.50	.20

Bootstrap Example

- Now create, many, say, *b*=10,000 virtual test sets *x*(*i*), each of size *n* = 10.
- To make each *x*(*i*), we randomly select a cell from row *x*, with replacement, 10 times:

	1	2	3	4	5	6	7	8	9	10	A%	B%	$\delta()$
x	AB	A~~B~~	AB	~~A~~B	A~~B~~	~~A~~B	A~~B~~	AB	~~A~~~~B~~	A~~B~~	.70	.50	.20
$x^{(1)}$	A~~B~~	AB	A~~B~~	~~A~~B	~~A~~B	A~~B~~	~~A~~B	AB	~~A~~~~B~~	AB	.60	.60	.00
$x^{(2)}$	A~~B~~	AB	~~A~~~~B~~	~~A~~B	~~A~~B	AB	~~A~~B	A~~B~~	AB	AB	.60	.70	-.10
…													
$x^{(b)}$													

Bootstrap Example

- Now we have a distribution. We can check how often A has an advantage on this test set.
  - If A is really better than B by about 0.2 then something like that value should show up often in our bootstraps.
  - However, if 0.2 is rare then A's advantage might be a fluke.
- Now assuming H_0, that means normally we expect δ(x')=0
- So just count how many times the δ(x') we found exceeds the expected 0 value by δ(x) or more:

$$\text{p-value}(x) = \sum_{i=1}^{b} \mathbb{1}\left(\delta(x^{(i)}) - \delta(x) \geq 0\right)$$

Bootstrap Example

- Alas, it's slightly more complicated. We didn't draw these samples from a distribution with 0 mean; we created them from the original test set *x*, which we know favors A
- So, to measure how surprising our observed δ(*x*) is, we actually compute the p-value by counting how often δ(*x*′) exceeds the expected value of δ(*x*) by δ(*x*) or more:

$$
\begin{aligned}
\text{p-value}(x) &= \sum_{i=1}^{b} \mathbb{1}\left(\delta(x^{(i)}) - \delta(x) \geq \delta(x)\right) \\
&= \sum_{i=1}^{b} \mathbb{1}\left(\delta(x^{(i)}) \geq 2\delta(x)\right)
\end{aligned}
$$

Bootstrap Example

- Suppose:
  - We have 10,000 test sets $x(i)$ and a threshold of .01
  - And in 47 of the test sets we find that $\delta(x(i)) \geq 2\delta(x)$
  - The resulting p-value is .0047
  - This is smaller than .01, indicating $\delta\ (x)$ is indeed sufficiently surprising
  - And we reject the null hypothesis and conclude *A* is better than *B*

Sounds Good

- This is a very effective way of determining progress in developing a system.
- However, it has some serious limitations in common scenarios
  - You may be comparing to someone’s published results without access to their system
  - Evaluation scripts are notoriously hard to get right. Don’t want to be comparing apples and oranges.
  - Test sets can get overused. Indirect contamination of the training by an overused test set.
  - Modern neural systems are stochastic -- different results from different training runs on the same data.

Moving on: Logistic Regression

- Naïve Bayes
- Logistic regression
  - Also known as log linear or maximum entropy (maxent) models

Logistic Regression Models

- Estimate $P(c|d)$ directly without Bayes using
  - A scoring function using
    - Features
    - Weights on those features
  - A classification strategy
  - A learning scheme

Scoring

- We’ll represent documents as sets of features.
- With each feature we’ll associate a weight (a real number).
- We’ll then assign a score to each document. For now let’s assume a binary classification problem. And that the score represents a document’s score as a positive example.

Scoring

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b$$

Note that this sum can be any arbitrarily large or small number.

Scoring

- Now we could just use that score and set a threshold such that if the score is > the threshold it's in, otherwise its out.
  - Think about spam detectors. They are essentially computing a score for how spam-like a message is. If the score exceeds a threshold we block it.

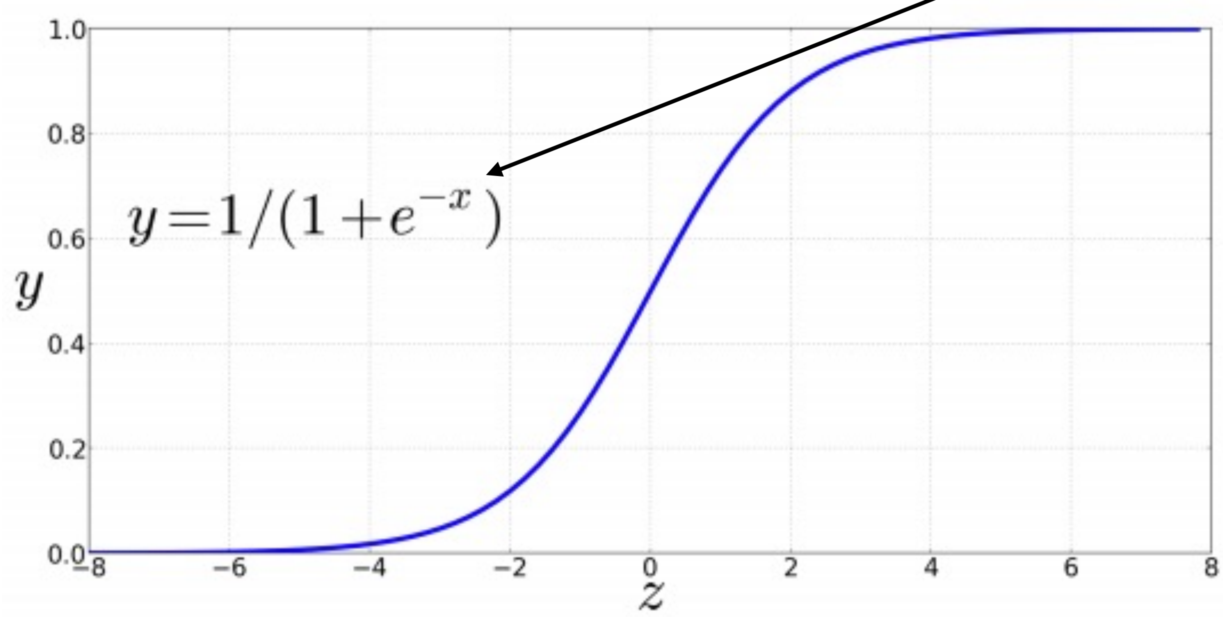
Better Scoring

- A better approach is to formulate that score as a probability P(class|document).
- This allows us to handle uncertainty in our classifications
  - Possibly useful in downstream applications
- And it facilitates learning

Better Scoring

- Squash the scores to between 0 and 1 using the sigmoid function

x is the old score



Better Scoring

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b$$

$$
\begin{aligned}
P(y=1) &= \sigma(w \cdot x + b) \\
&= \frac{1}{1+e^{-(w \cdot x+b)}}
\end{aligned}
$$

$$
\begin{aligned}
P(y=0) &= 1 - \sigma(w \cdot x + b) \\
&= 1 - \frac{1}{1+e^{-(w \cdot x+b)}} \\
&= \frac{e^{-(w \cdot x+b)}}{1+e^{-(w \cdot x+b)}}
\end{aligned}
$$

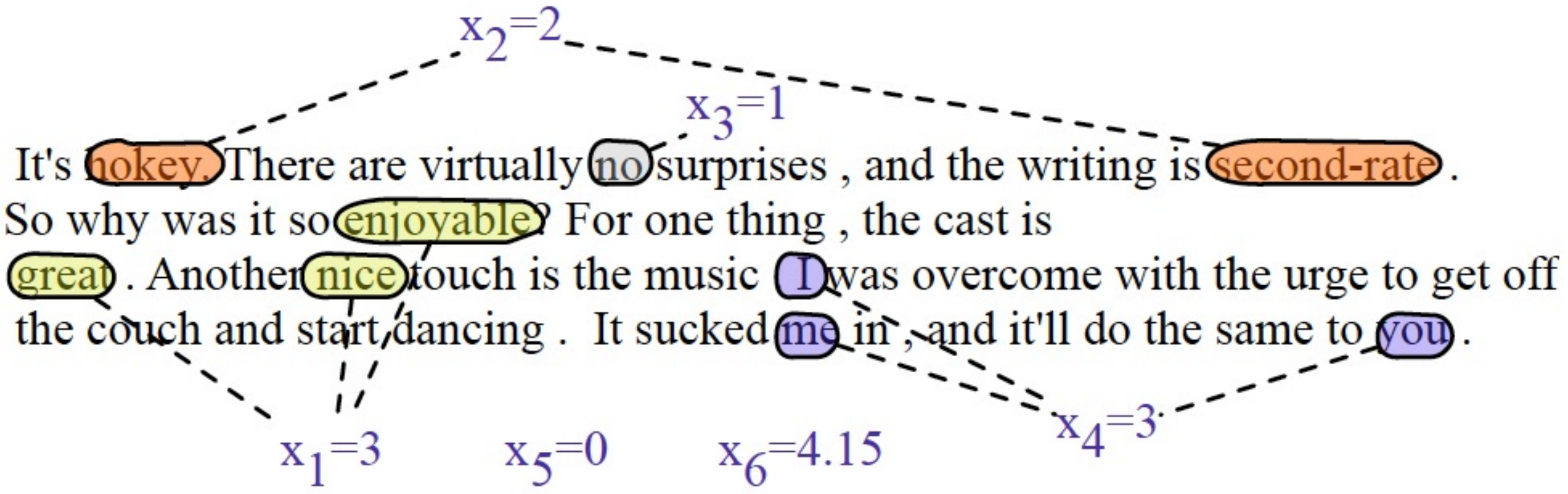
Features

- The kind of features used in NLP-oriented ML-based classifier systems are
  1. Easily extracted from a text
  2. Hand-crafted based on domain knowledge and data analysis
  3. And we have lots and lots of them

Sentiment Features

- Given lists of positive and negative words
  - Called a ”sentiment lexicon”
    - The count of each type in a review
- The presence of “no” (and other negations) in the review
- Use of pronouns
- Use of punctuation (like !!!)
- Review length

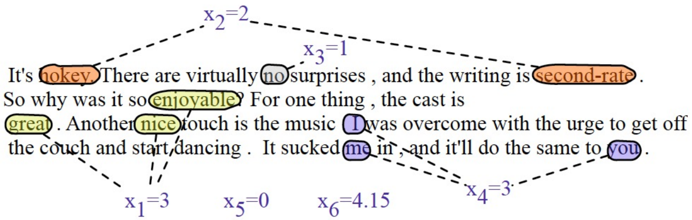
Sentiment Features



Sentiment Features

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

Scoring

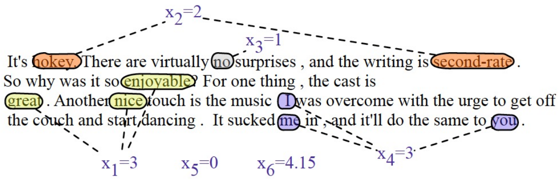


[3, 2, 1, 3, 0, 4.15]

Now assume we have already have weights for each of these features.

2.5, -5.0, -1.2, 0.5, 2.0, 0.7

Scoring



[3,2,1,3,0,4.15]

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.15] + 0.1) \\
&= \sigma(.805) \\
&= 0.69 \qquad (5.6) \\
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.31
\end{aligned}
$$

Multiclass Models

- While useful, binary classification isn't the typical use case in NLP. Rather we're classifying and object into one of *N* categories.
- We'd like a model that provides a probability distribution over the categories given an object.

Multiclass Models

- Multinomial logistic regression
- We'll still assume we have objects represented as vectors of features, and weights on the features, and a scoring function.

Softmax Classification

- Given an object x, the softmax function gives us the probability distribution over the classes.
- We can take the argmax and go with that or we can simply use the distribution as the answer (passing it on for further processing).

Softmax

This gives us a score for a single class *c*.

$$p(y=c|x) = \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}$$

Result is a normalized probability distribution over the classes.

This sums the scores for all the classes.

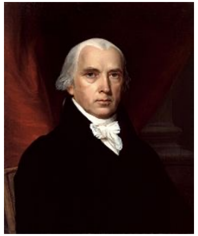
Note that the feature weights and bias term are now indexed per class.

Federalist Example

$$p(y=c|x) = \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}$$

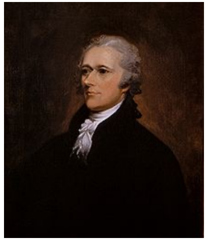
.00039

P(Jay | doc) = .00039/(.00039+.0409+.0000017)
= .00946



.0409

P(Madison | doc) = .0409/(.00039+.0409+.0000017)
= .99



.0000017

P(Hamilton | doc) = .0000017/(.00039+.0409+.0000017)
= .000041

Weights

- So… where do those weights come from?

Weights

- So… where do those weights come from?
- We'll learn the weights from a training set of documents already labeled with the right answer.
  - We'll use 1 and 0 as labels to stand for the right answer (positive or negative)
  - The system answers will be between 0 and 1

Weights

- To learn the weights, we need some measure of how well (or badly) we're doing with a current set of weights.
- We'll call that measure a *Loss Function*
- The lower the loss the better we're doing
  - We want to minimize the loss

Loss Functions

- There are lots of ways to evaluate how well we're doing on a test/validation set.
  - Accuracy, F1, Precision, recall.
    - All discrete
  - How close are our answers are to the correct answers?
    - What does close even mean?
  - What's the nature of the overall loss over the entire training/test data

Loss Functions

- What we want is a function that tells us how well our model is doing.
- And does it in a way that can be used to guide the training process

Cross Entropy Loss

- Let's start with the following notion

the probability assigned by a model <u>to the correct answer</u>

  - In the binary case, the correct answer is always either 1 or 0.
  - The model output is a number between 0 and 1. That represents the probability that the item belongs to the class.
    - By convention, the probability given by the model is the probability that the doc belongs to class 1 (or the positive class)

Cross Entropy Loss

- If the correct answer for an example is 1 and the system output is .7 then the probability assigned to the correct class is
  - ?
- If the system answer is .8 and the correct answer is 0 then
  - ?

Cross Entropy Loss

- If the correct answer for an example is 1 and the system output is .7 then
  - 0.7
- If the system answer is .8 and the correct answer is 0 then
  - 0.2

Cross Entropy Loss

- If the correct answer for an example is 1 and the system output is .2 then
  - ?
- If the system answer is .1 and the correct answer is 0 then
  - ?

Cross Entropy Loss

- If the correct answer for an example is 1 and the system output is .2 then
  - 0.2
- If the system output is 0.1 and the correct answer is 0 then
  - 0.9

Cross Entropy Loss

$$p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$

y is the correct answer.

y_hat is the system answer.

Logs

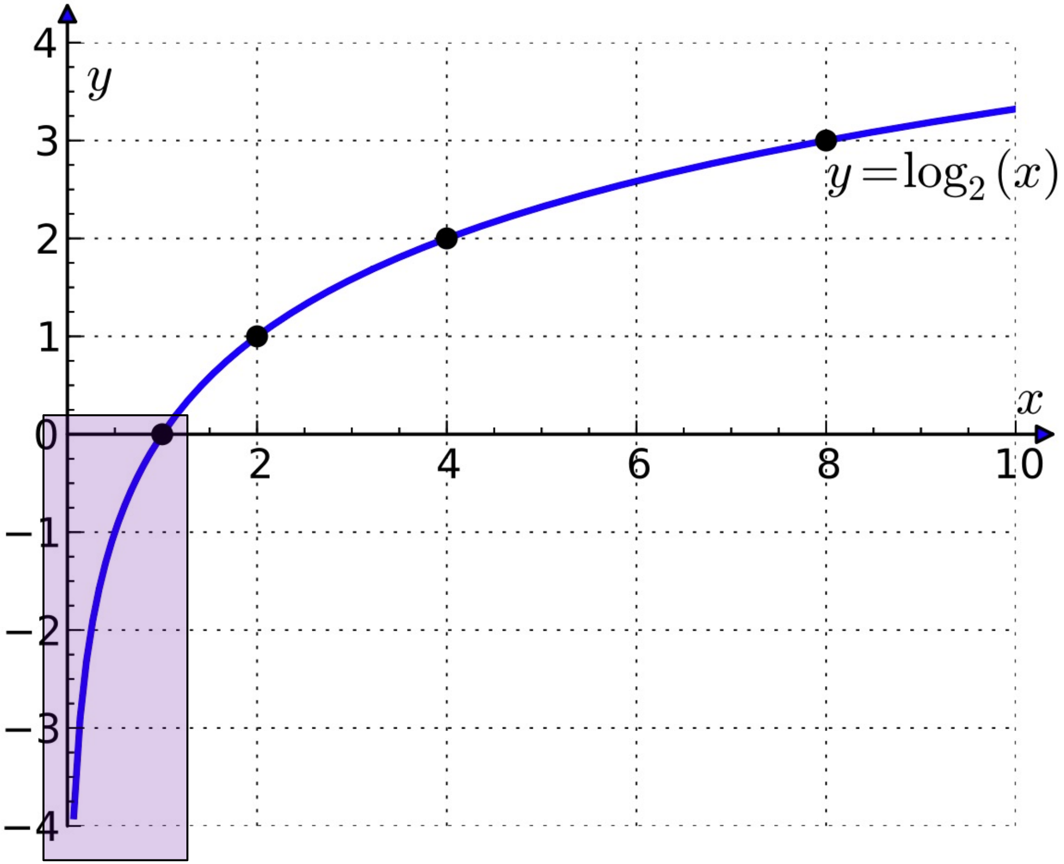
- Probabilities are the opposite of what we want for a loss.
  - We want bad performance to have high loss and good performance to have low loss.
- So, we'll take the negative of the log of the probability assigned to the correct answer as the loss

Logs

Log of a value between 0 and 1 ranges from -infinity to 0.

Taking the negative of that it ranges from infinity to 0.

Low prob → large value
High prob → small value



Cross Entropy Loss

$$p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$

$$\begin{aligned}\log p(y|x) &= \log\left[\hat{y}^y (1-\hat{y})^{1-y}\right] \\ &= y\log\hat{y} + (1-y)\log(1-\hat{y})\end{aligned}$$

$$L_{CE}(w,b) = -[y\log\sigma(w\cdot x+b) + (1-y)\log(1-\sigma(w\cdot x+b))]$$

Negative of the log probability the model assigns to the correct answer.

Learning

- We want to find the weights that minimize the average loss across an entire training set.

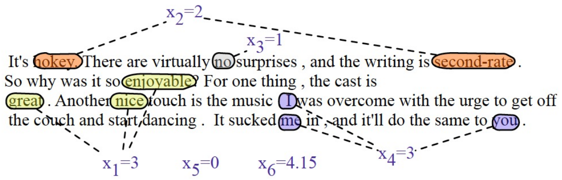
$$
\begin{aligned}
Cost(w,b) &= \frac{1}{m}\sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)}) \\
&= -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)}) \log \left(1 - \sigma(w \cdot x^{(i)} + b)\right)
\end{aligned}
$$

Learning

- We'll do this by starting with a random set of weights and then iteratively updating those weights to lower this cost.

$$
\begin{aligned}
Cost(w,b) &= \frac{1}{m}\sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)}) \\
&= -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)}) \log \left(1 - \sigma(w \cdot x^{(i)} + b)\right)
\end{aligned}
$$

Scoring



[3,2,1,3,0,4.15]

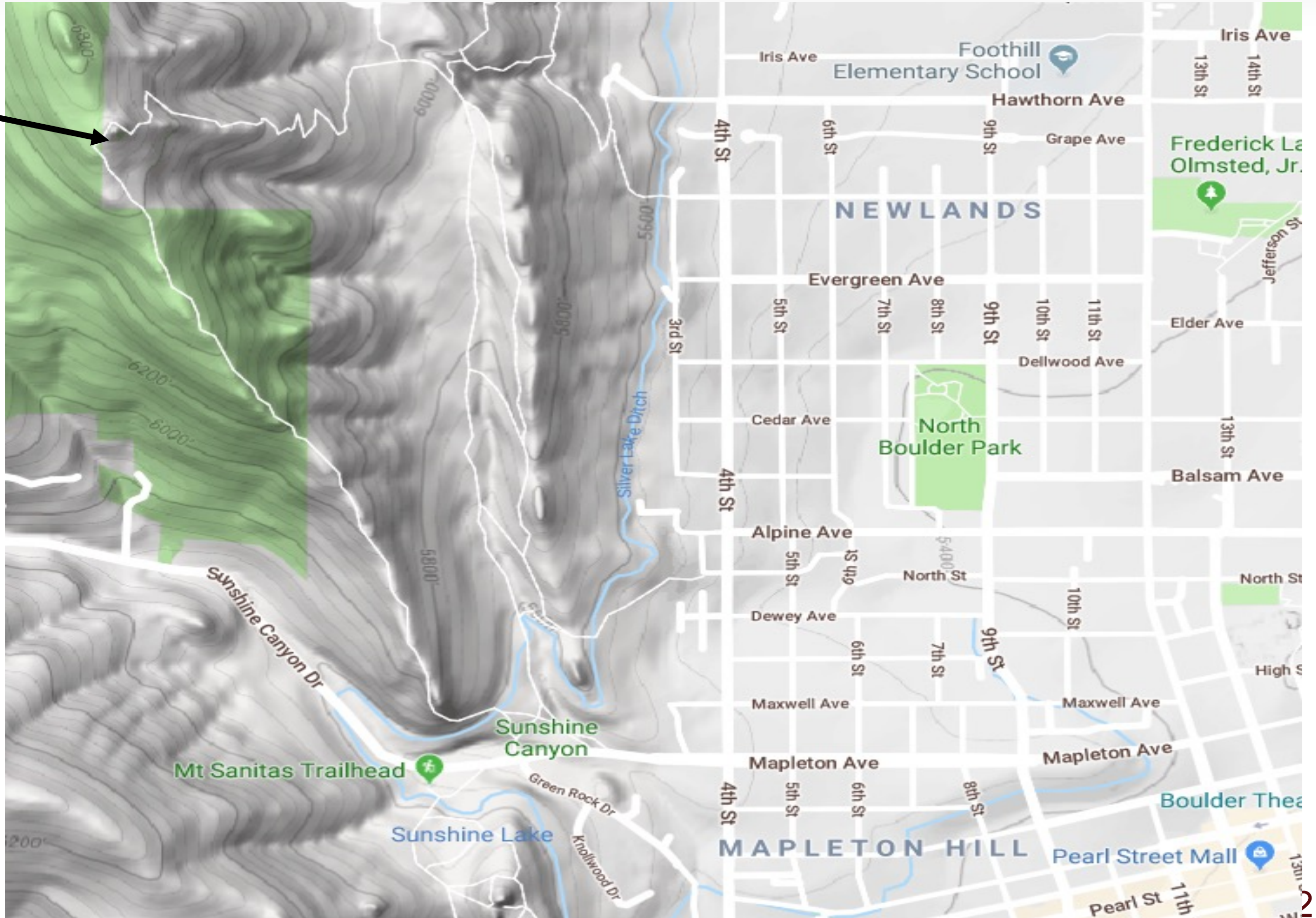
$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3,2,1,3,0,4.15] + 0.1) \\
&= \sigma(.805) \\
&= 0.69 \qquad (5.6) \\
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.31
\end{aligned}
$$

Learning

$$w_{t+1} = w_t - \mu\Delta$$

Motivation

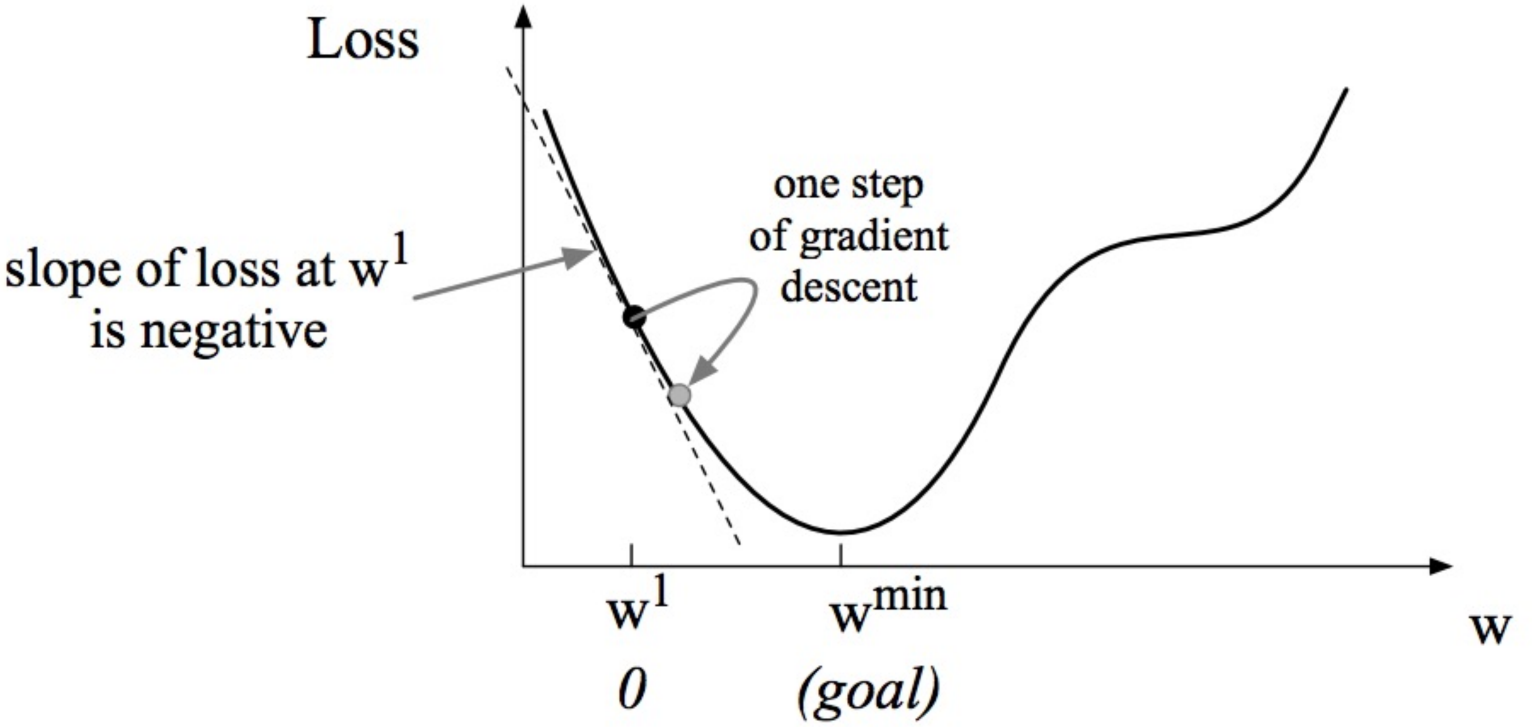
This is you.
Find the fastest
way down.



Derivatives

- Fortunately, basic calculus tells us how to do that.
- The derivative of the loss function with respect to the weights tells us the direction and magnitude of change we should make to each weight.

Single Weight



Partial Derivative

- Of course, in real applications we have many features/weights not just one.
- So we need the vector of the partial derivatives of the loss wrt the weights
  - Call that the gradient

Partial Derivative

$$L_{CE}(w,b) = -[y\log\sigma(w\cdot x+b)+(1-y)\log(1-\sigma(w\cdot x+b))]$$

$$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [\sigma(w\cdot x+b)-y]x_j$$

If we have *n* weights, we end up with *n* partial derivatives. The vector consisting of those derivatives is called the gradient. We'll use the gradient to update all the weights.

CE Loss Partial Derivative

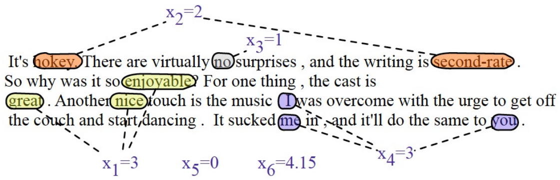
- Looks like gobbledygook, but it matches our earlier intuition about the loss and "credit/blame" assignment for the error

$$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [\underbrace{\sigma(w\cdot x+b)}_{\text{Computed answer}} - \underbrace{y}_{\text{right answer}}]x_j$$

Take the difference

Multiply by the value of the input feature for the weight being adjusted

Scoring



[3,2,1,3,0,4.15]

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.15] + 0.1) \\
&= \sigma(.805) \\
&= 0.69 \qquad (5.6) \\
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.31
\end{aligned}
$$

Updates

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.15] + 0.1) \\
&= \sigma(.805) \\
&= 0.69 \qquad (5.6) \\
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.31
\end{aligned}
$$

$$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

$$w_{t+1} = w_t - \mu\Delta$$

SGD

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ

  # where: L is the loss function

  #   f is a function parameterized by θ

  #   x is the set of training inputs $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$

  #   y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$

$\theta \leftarrow 0$

repeat T times

  For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)

  Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output $\hat{y}$?

  Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$) from the true output $y^{(i)}$?

  $g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss ?

  $\theta \leftarrow \theta - \eta\, g$ # go the other way instead

return θ

Optimization

- In practice, that can be slow to converge because the algorithm can either be taking steps
  - That are too small and hence take us too long to get where we’re going
  - Or too large which leads us to overshoot the target and wander around too much
- Fortunately, you don’t have to worry about this. Lots of packages available where you need to specify L and L’ and you’re done.