

SGD, Data Prep, and other Practicalities

Lecture 5

Last Time

- ***Logistic regression*** is a hypothesis class for predicting the probability of a discrete class label, which is log proportional to linear combination of attributes
 - We can generalize it to the multiclass case ($k > 2$)
 - But no closed-form solution for ERM
- Textbook: sections 9.3

This Class

- How do we find an ERM for logistic regression?
- How do we apply logistic regression to real-world problems that have
 - Lots of data?
 - Features of different types?
- Textbook: sections 12.1.1, 14.0, 14.1.0, 14.3.0, 14.5.1

Gradient Descent

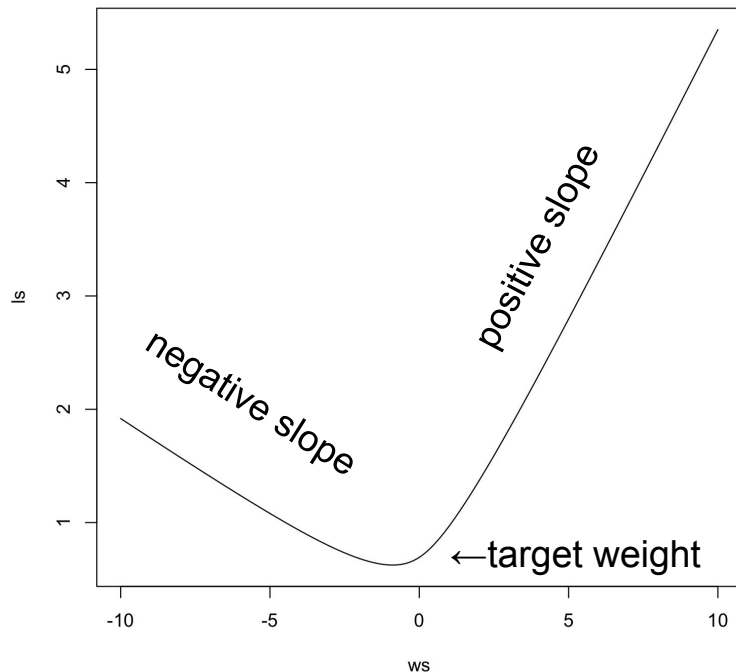
Gradient Descent

Our function has a minimum and the derivatives point away from it.

$$w_{st} \leftarrow w_{st} - \alpha \frac{\partial L_S(h_{\mathbf{w}})}{\partial w_{st}}$$

α is a user-specified value, i.e., a hyperparameter, called the step size

Repeat the update until convergence, i.e., the loss doesn't change significantly



Question



We'll Have to Wait for the Question



How do we Set the Step Size?

- Since the loss for logistic regression is convex, okay to try different values and see which leads to lowest training loss in reasonable time.
- Common values range from $1e^{-3}$ to $1e^{-6}$
- Some optimizers automatically tune step size, such as AdaGrad and Adam

Gradient Descent in Higher Dimensions

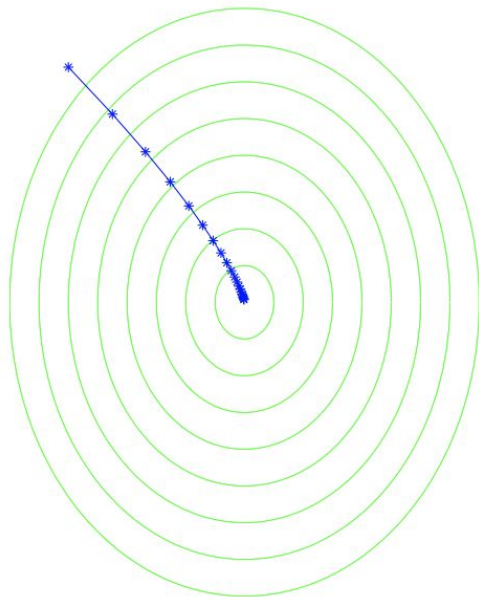
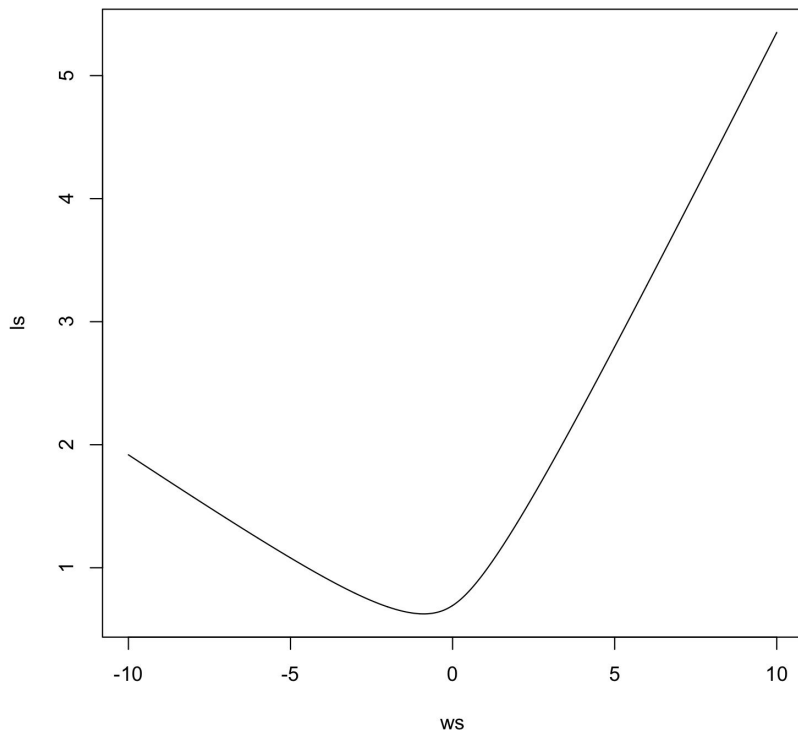


Figure 14.1 An illustration of the gradient descent algorithm. The function to be minimized is $1.25(x_1 + 6)^2 + (x_2 - 8)^2$.

Understanding Machine Learning.
Shalev-Shwartz and Ben-David, 2014.

How Many Examples Do We Need for an Update?



- Suppose that we're here and we start computing the gradient of the loss for examples in our training data
- If the first 9 out of 10 examples have positive gradient, do we really need to go over all the data ?

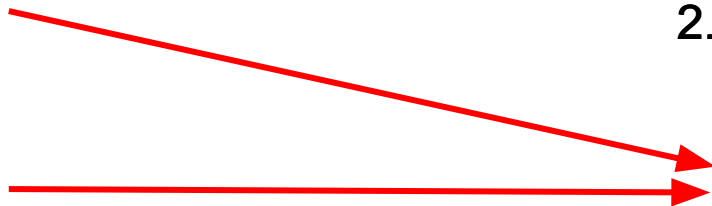
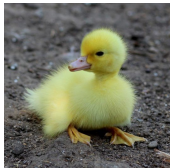
Stochastic Gradient Descent

Stochastic Gradient Descent

- Idea: don't compute the gradient of $L_S(h_{\mathbf{w}})$ exactly, instead use one or a batch of examples to estimate the gradient of $L_{\mathcal{D}}(h_{\mathbf{w}})$
- It's noisy, but correct in expectation. Advantages:
 1. Faster gradient computation
 2. Less overfitting

Example: Stochastic Gradient Descent

Training Data S

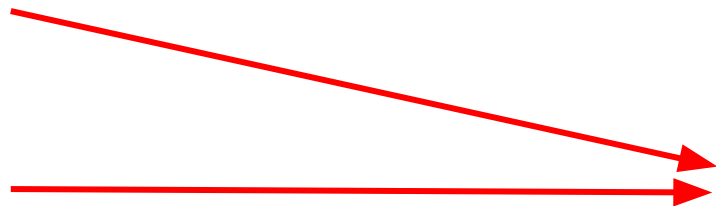
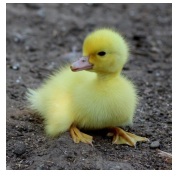


1. Initialize weights randomly
2. Create a “batch” S' from the training data of size b
3. Take step with gradient of empirical risk on S'
4. Go back to 3 with next batch until convergence



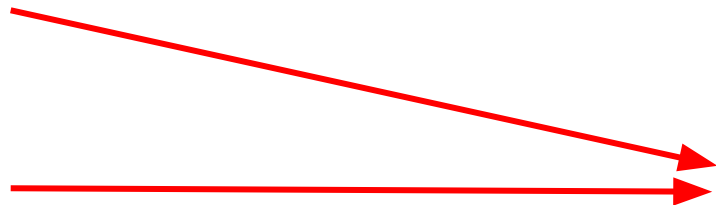
Example: Stochastic Gradient Descent

Training Data S



Batch 1

Batch 4...

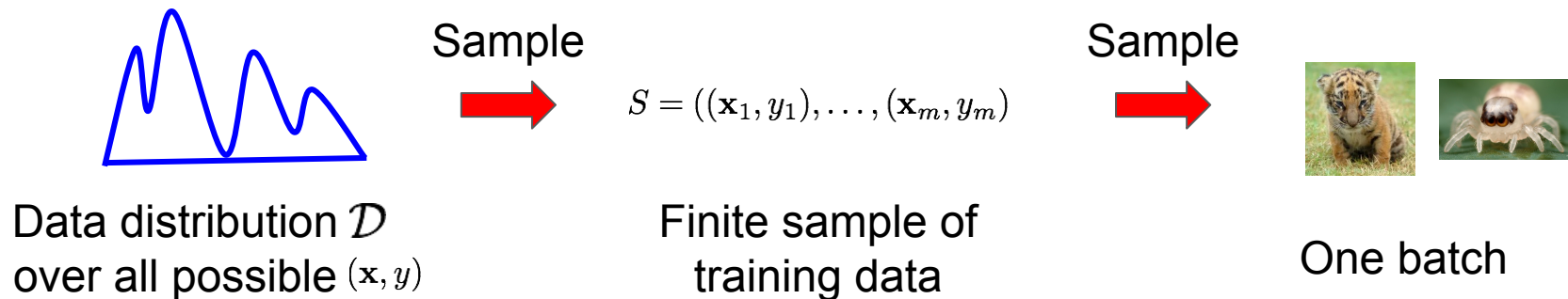


Batch 2



Batch 3

Why Does it Work?



- If there is no bias in either sampling stage, then each gradient of the empirical risk we compute has same expected value as that of true risk
- Once we repeat training examples, batches are no longer independent. In practice, we just shuffle the training data and it works out fine

Why Does it Work?

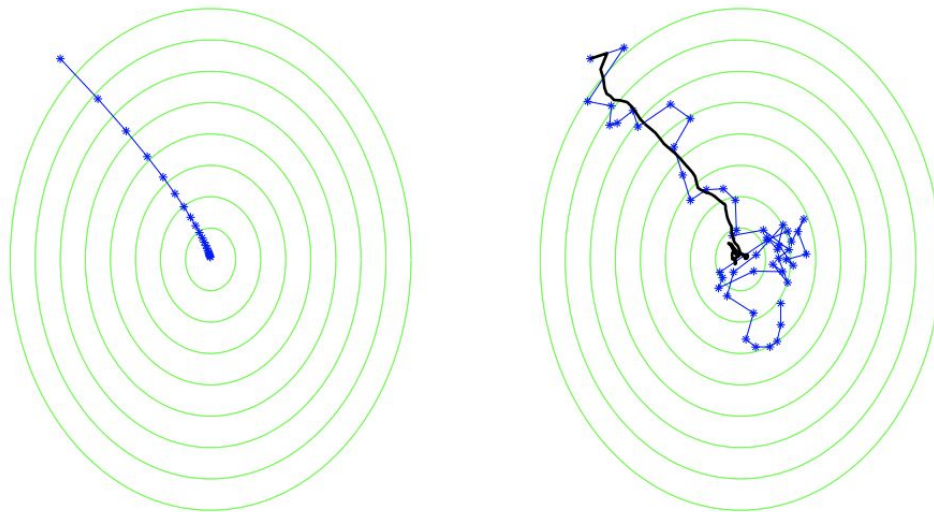


Figure 14.3 An illustration of the gradient descent algorithm (left) and the stochastic gradient descent algorithm (right). The function to be minimized is $1.25(x + 6)^2 + (y - 8)^2$. For the stochastic case, the black line depicts the averaged value of \mathbf{w} .

Stochastic Gradient Descent for Logistic Regression

Inputs: training examples S , step size α , batch size $b < |S|$

Initialize $\mathbf{w} \in \mathbb{R}^{k \times d}$ randomly

converged \leftarrow False

while !converged:

 Shuffle S

for $i = 0 \dots \lceil |S| / b \rceil - 1$:

$S' \leftarrow S[i \cdot b : (i + 1) \cdot b]$

$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla L_{S'}(h_{\mathbf{w}})$

 converged \leftarrow check_convergence(S, \mathbf{w})

return

Some Terminology

Inputs: training examples S , step size α , batch size $b < |S|$

Initialize $\mathbf{w} \in \mathbb{R}^{k \times d}$ randomly

converged \leftarrow False

while !converged:

 Shuffle S

for $i = 0 \dots \lceil |S| / b \rceil - 1$:

$S' \leftarrow S[i \cdot b : (i + 1) \cdot b]$

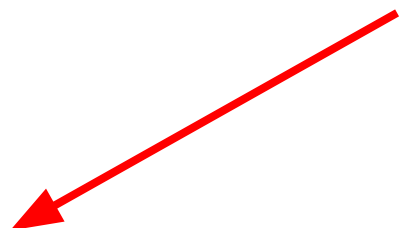
$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla L_{S'}(h_{\mathbf{w}})$

 converged \leftarrow check_convergence(S, \mathbf{w})

return

One “batch”

- Or one “step”
- Or one “update”



Some Terminology

Inputs: training examples S , step size α , batch size $b < |S|$

Initialize $\mathbf{w} \in \mathbb{R}^{k \times d}$ randomly

converged \leftarrow False

while !converged:

 Shuffle S

for $i = 0 \dots \lceil |S| / b \rceil - 1$:

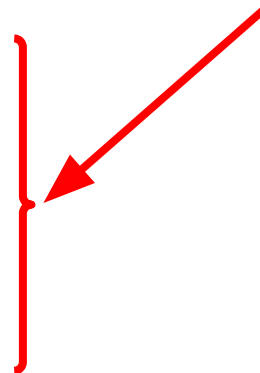
$S' \leftarrow S[i \cdot b : (i + 1) \cdot b]$

$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla L_{S'}(h_{\mathbf{w}})$

 converged \leftarrow check_convergence(S, \mathbf{w})

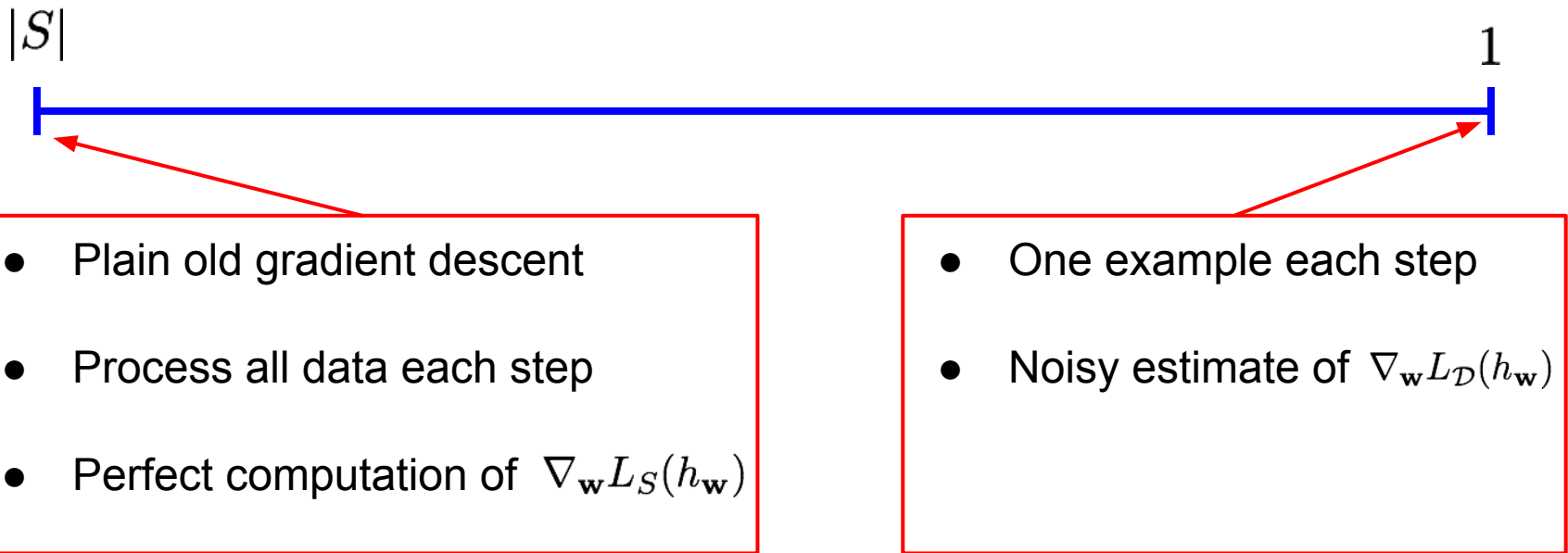
return

One “epoch”



Choosing a Batch Size

- What is the relationship between batch size and the behavior of SGD?



Question



We'll Have to Wait for the Question



Checking for Convergence

Inputs: training examples S , step size α , batch size $b < |S|$

Initialize $\mathbf{w} \in \mathbb{R}^{k \times d}$ randomly

converged \leftarrow False

while !converged:

 Shuffle S

for $i = 0 \dots \lceil |S| / b \rceil - 1$:

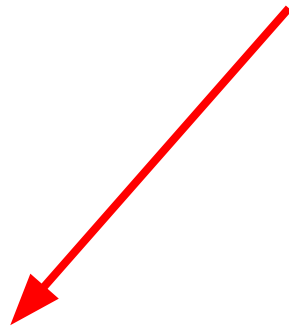
$S' \leftarrow S[i \cdot b : (i + 1) \cdot b]$

$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla L_{S'}(h_{\mathbf{w}})$

 converged \leftarrow check_convergence(S, \mathbf{w})

return

How do we
do this?



Ideas

- Check how much the weights have changed?
 - Might waste time when empirical risk has wide, flat basin
- Check how much the empirical risk has changed?
 - We can periodically compute the empirical risk
 - Check for small changes? Still have to define “small”
 - Check if empirical has improved in last few epochs?
- For homework, it's sufficient to stop if empirical risk is no less than previous epoch's empirical risk

$$\|\mathbf{w}^{(t+1)} - \mathbf{w}^{(t)}\|_2 < \epsilon$$

$$|L_S(h_{\mathbf{w}^{(t+1)}}) - L_S(h_{\mathbf{w}^{(t)}})| < \epsilon$$

$$L_S(h_{\mathbf{w}^{(t+1)}}) > L_S(h_{\mathbf{w}^{(t)}})$$

Representing Data as Vectors

What kinds of data can we represent?

| | Num. Eyes | Num. Legs | Num. Fins |
|--|-----------|-----------|-----------|
|  Tiger | 2 | 4 | 0 |
|  Spider | 8 | 8 | 0 |
|  Shark | 2 | 0 | 2 |
|  Snake | 2 | 0 | 0 |



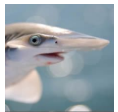
So far, numeric data has been easy to represent as vectors

What kinds of data can we represent?

| | Stripes | Color | Size |
|--|---------|--------|--------|
|  Tiger | True | Orange | Large |
|  Spider | True | White | Tiny |
|  Shark | False | Gray | Medium |
|  Snake | False | Green | Medium |

What about other kinds of data?

Binary Features

| | Stripes |
|--|---------|
|  Tiger | True |
|  Spider | True |
|  Shark | False |
|  Snake | False |

- We've already seen a preview of this with halfspaces
- Just assign one dimension to the feature and map True to 1 and False to 0

Categorical Features

| | Color |
|--|--------|
|  Tiger | Orange |
|  Spider | White |
|  Shark | Gray |
|  Snake | Green |



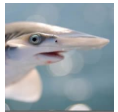

- Why handle these differently? (What would happen if we handled them like numeric attributes?)
- Map each value to a dimension and set that dimension to 1 if an example has that value
- “One-hot encoding”

Ordinal Features

| | Size |
|--|--------|
|  Tiger | Large |
|  Spider | Tiny |
|  Shark | Medium |
|  Snake | Medium |

- Now there is some ordering between the values!
- But is the difference between Tiny and Medium the same as between Medium and Large?
- Assign each value to a dimension, and set each dimension to 1 up to and including the largest value

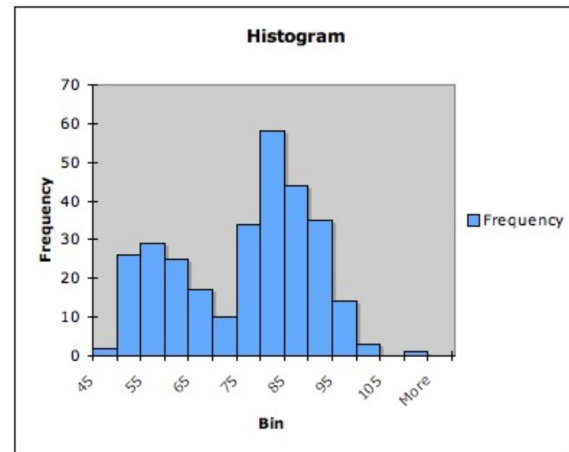
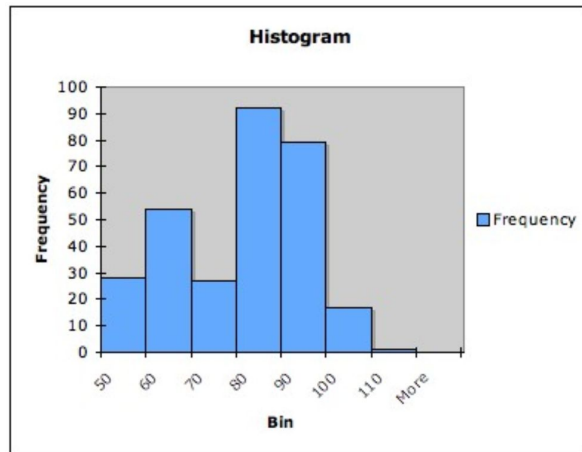
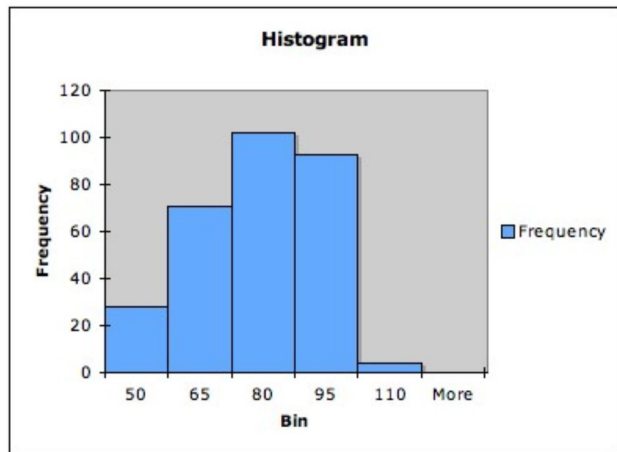
Discretization

| | | Size | Weight |
|---|--------|--------|--------|
|  | Tiger | Large | 408 lb |
|  | Spider | Tiny | 3 oz |
|  | Shark | Medium | 200 lb |
|  | Snake | Medium | 50 lb |

- How do you arrive at these feature labels?
- Discretization: splitting up a continuous variable into bins or categories
- What information is lost?
 - e.g., distance: a shark is much larger than a snake, but that is no longer reflected

More Limitations of Features Engineering

- Loss of information might lead to misleading results
- Histograms of times between successive eruptions of Old Faithful





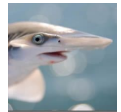
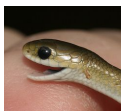
Potential Consequences

- Predicting the next Yellowstone volcano eruption?
- What happens when we discretize human features?
 - Binarizing gender identities
 - Categorizing mixed race people
 - How do we discretize disabilities? How does that affect accessibility considerations?

Normalizing Data

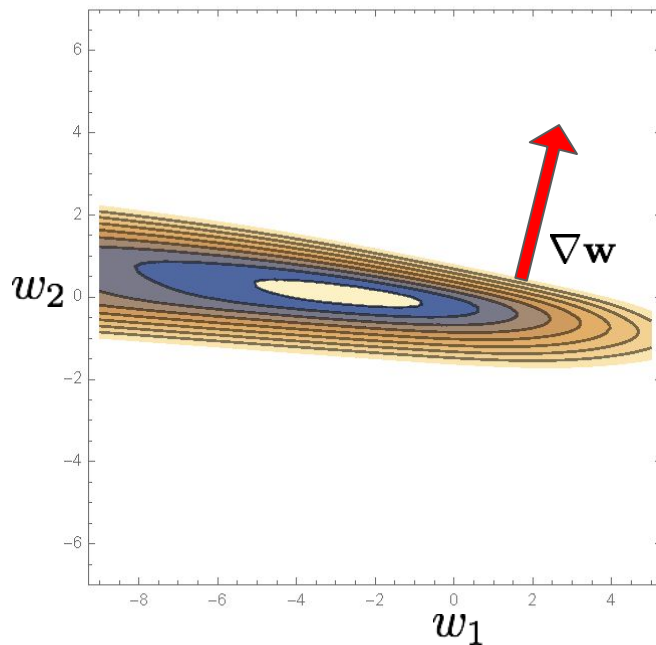
Multi-Scale Data

- Let's look at the log loss for data with attributes on different scales

| | Fuzzy | Num. Legs | Weight (oz) |
|--|-------|-----------|-------------|
|  Tiger | 1 | 4 | 6,528 |
|  Spider | 1 | 8 | 3 |
|  Shark | 0 | 0 | 3,200 |
|  Snake | 0 | 0 | 800 |

Empirical Log Loss Contour Plot

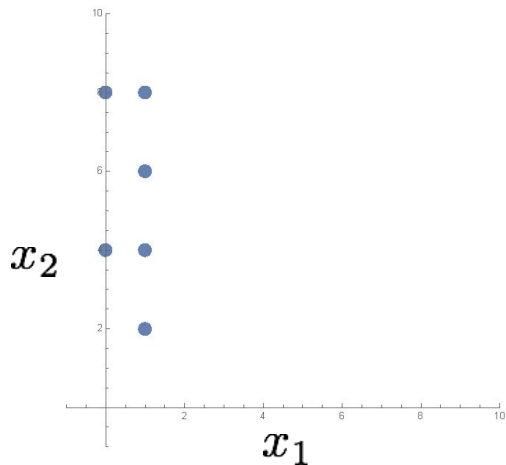
- $x_1 \in [0, 1]$ And $x_2 \in [2, 8]$
- Example: gradient at $\mathbf{w} = (1, 1)$ is $(3.96, 19.88)$
- If step size is high, overshooting in w_2
- If step size is low, undershooting in w_1



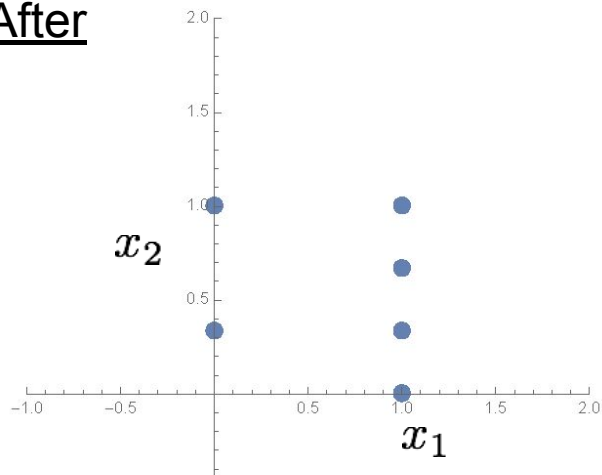
Min-Max Scaling

- Simple fix in many cases. For each attribute
 - 1) Subtract the minimum value
 - 2) Divide by (max - min)
- Helper class: [Scikit-Learn MinMaxScaler](#)

Before

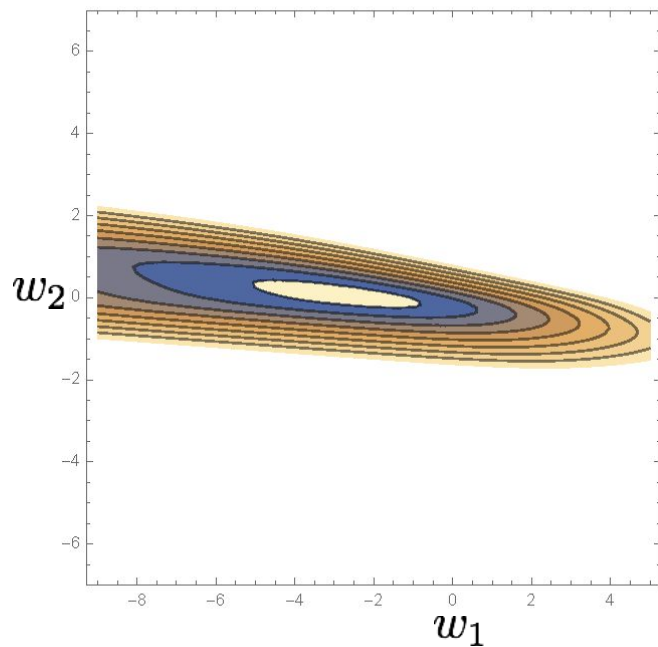


After

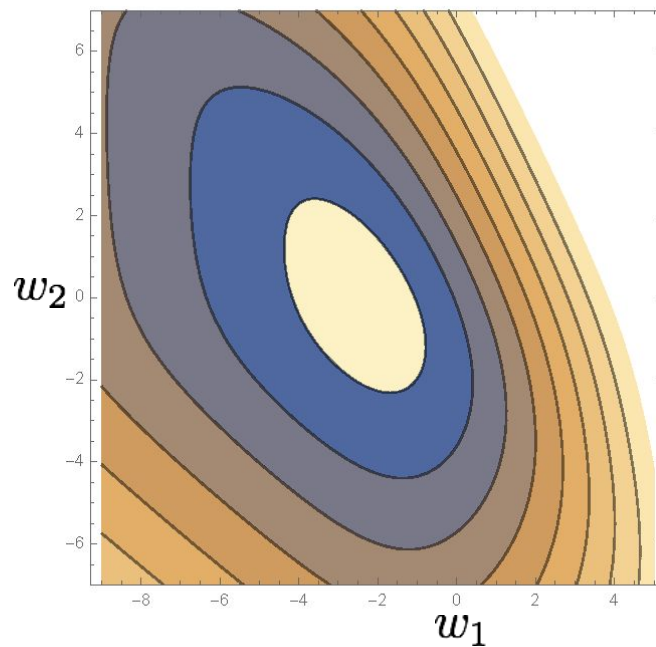


Empirical Log Loss Contour Plot

Before



After



The Most Important Things

- Stochastic gradient descent (SGD) is a variant of gradient descent that is faster and more scalable.
 - Just like gradient descent, except we compute the gradient on small batches of training data
- In addition to step size, other hyperparameters include batch size and the convergence criterion
- How we represent our data affects which hypotheses we learn
- Normalizing your data is often needed for best performance
- Textbook: sections 12.1.1, 14.0, 14.1.0, 14.3.0, 14.5.1

Next Class

- Now that we have some hypothesis classes in our toolbox, what can we prove formally about machine learning algorithms that use the ERM principle?
- Textbook: chapters 2.3, 3