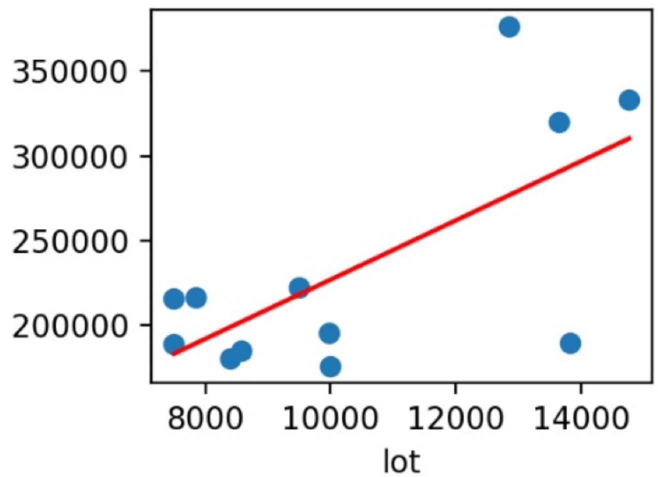
CMSC 478
Lecture 3

KMA Solaiman

Supervised Learning: Classification, Perceptrons

Some slides are slightly adapted from Chris Re´, Stanford ML

Visual version of linear regression: Learning



Let $h_\theta(x) = \sum_{j=0}^{d} \theta_j x_j$ want to choose θ so that $h_\theta(x) \approx y$. One popular idea called **least squares**

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 .$$

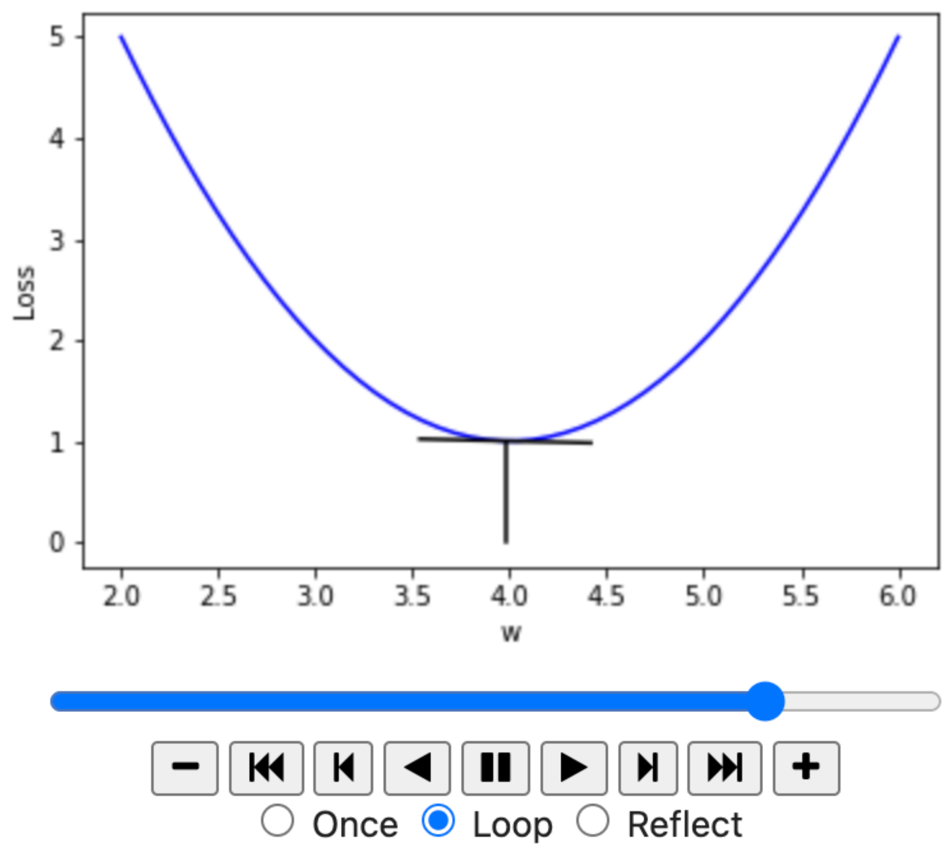
Choose

$$\theta = \underset{\theta}{\operatorname{argmin}}\, J(\theta).$$

Solving the least squares optimization problem.

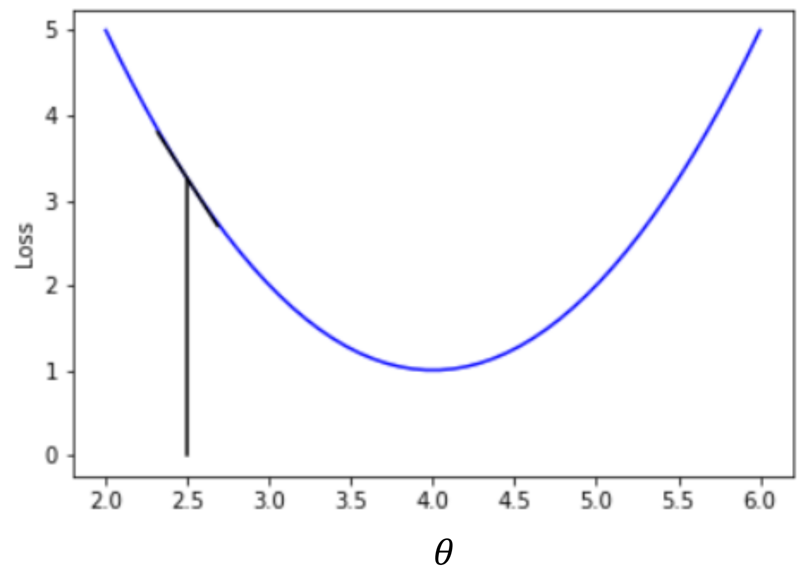
Gradient Descent

Animation



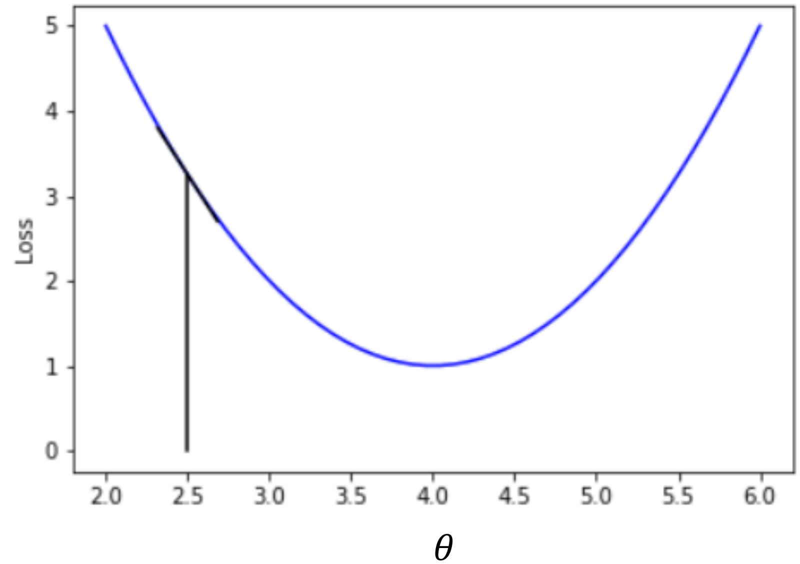
Gradient Descent

- $\mathcal{J}(\theta) = (\theta - 4)^2 + 1$
- Find the weight (value of θ) that minimizes the loss $\mathcal{J}$
- $\mathcal{J}'(\theta) = ?$
- θ = 2.5
- given the current value of w, adjusting θ by an amount that has the negative of the sign of $\mathcal{J}'(\theta)$ leads to a smaller value of $\mathcal{J}$.



Gradient Descent

- $\mathcal{J}(\theta) = (\theta - 4)^2 + 1$
- Find the weight (value of θ) that minimizes the loss J
- $\mathcal{J}'(\theta) = ?$
- θ = 2.5
- given the current value of w, adjusting θ by an amount that has the negative of the sign of $\mathcal{J}'(\theta)$ leads to a smaller value of $\mathcal{J}$.



$$\theta = \theta - \alpha * \mathcal{J}'(\theta)$$

Gradient Descent

	size	bedrooms	lot size
$x^{(1)}$	2104	4	45k
$x^{(2)}$	2500	3	30k

	Price
$y^{(1)}$	400
$y^{(2)}$	900

What's a prediction here?

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3.$$

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2.$$

$$\theta^{(0)} = 0$$

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha \frac{\partial}{\partial \theta_j} J(\theta^{(t)}) \qquad \text{for } j = 0, \ldots, d.$$

Gradient Descent Computation

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha \frac{\partial}{\partial \theta_j} J(\theta^{(t)}) \text{ for } j = 0, \ldots, d.$$

Note that α is called the **learning rate** or **step size**.

Let's compute the derivatives...

$$\begin{aligned} \frac{\partial}{\partial \theta_j} J(\theta^{(t)}) &= \sum_{i=1}^{n} \frac{1}{2} \frac{\partial}{\partial \theta_j} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \\ &= \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \frac{\partial}{\partial \theta_j} h_\theta(x^{(i)}) \end{aligned}$$

Gradient Descent Computation

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha \frac{\partial}{\partial \theta_j} J(\theta^{(t)}) \text{ for } j = 0, \ldots, d.$$

Note that α is called the **learning rate** or **step size**.

Let's compute the derivatives. . .

$$\begin{aligned} \frac{\partial}{\partial \theta_j} J(\theta^{(t)}) &= \sum_{i=1}^{n} \frac{1}{2} \frac{\partial}{\partial \theta_j} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \\ &= \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \frac{\partial}{\partial \theta_j} h_\theta(x^{(i)}) \end{aligned}$$

For our *particular* h_θ we have:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_d x_d \text{ so } \frac{\partial}{\partial \theta_j} h_\theta(x) = x_j$$

Gradient Descent Computation

Thus, our update rule for component j can be written:

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}.$$

Gradient Descent Computation

Thus, our update rule for component j can be written:

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}.$$

We write this in *vector notation* for $j = 0, \ldots, d$ as:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}.$$

Saves us a lot of writing! And easier to understand . . . eventually.

Batch Versus Stochastic Minibatch: Motivation

Consider our update rule:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}.$$

- A single update, our rule examines *all* n data points.

Batch Versus Stochastic Minibatch: Motivation

Consider our update rule:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}.$$

- A single update, our rule examines *all* n data points.
- In some modern applications (more later) n may be in the billions or trillions!
  - E.g., we try to "predict" every word on the web.

Batch Versus Stochastic Minibatch: Motivation

Consider our update rule:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}.$$

- A single update, our rule examines *all* n data points.
- In some modern applications (more later) n may be in the billions or trillions!
  - E.g., we try to "predict" every word on the web.
- **Idea** Sample a few points (maybe even just one!) to *approximate* the gradient called **Stochastic Gradient** (SGD).
  - SGD is the workhorse of modern ML, e.g., pytorch and tensorflow.

Stochastic Minibatch

- We randomly select a **batch** of $B \subseteq \{1, \ldots, n\}$ where $|B| < n$.
- We approximate the gradient using just those B points as follows (vs. gradient descent)

$$\frac{1}{|B|} \sum_{j \in B} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)} \text{ v.s. } \frac{1}{n} \sum_{j=1}^{n} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)}.$$

Stochastic Minibatch

- We randomly select a **batch** of $B \subseteq \{1, \ldots, n\}$ where $|B| < n$.
- We approximate the gradient using just those B points as follows (vs. gradient descent)

$$\frac{1}{|B|}\sum_{j \in B}\left(h_\theta(x^{(j)}) - y^{(j)}\right)x^{(j)} \text{ v.s. } \frac{1}{n}\sum_{j=1}^{n}\left(h_\theta(x^{(j)}) - y^{(j)}\right)x^{(j)}.$$

- So our update rule for SGD is:

All minibatches are used for each iteration, or epoch and then start the next one

$$\theta^{(t+1)} = \theta^{(t)} - \alpha_B \sum_{j \in B}\left(h_\theta(x^{(j)}) - y^{(j)}\right)x^{(j)}.$$

- NB: scaling of $|B|$ versus n is “hidden” inside choice of α_B.

Stochastic Minibatch vs. Gradient Descent

- Recall our rule B points as follows:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha_B \sum_{j \in B} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)}.$$

- If $|B| = \{1, \ldots, n\}$ (the whole set), then they coincide.
- Smaller B implies a lower quality approximation of the gradient (higher variance).
- Nevertheless, it may actually converge faster! (Case where the dataset has many copies of the same point–extreme, but lots of redundancy)

Stochastic Minibatch vs. Gradient Descent

- Recall our rule B points as follows:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha_B \sum_{j \in B} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)}.$$

- If $|B| = \{1, \dots, n\}$ (the whole set), then they coincide.
- Smaller B implies a lower quality approximation of the gradient (higher variance).
- Nevertheless, it may actually converge faster! (Case where the dataset has many copies of the same point–extreme, but lots of redundancy)
- In practice, choose B proportional to what works well on modern parallel hardware (GPUs).

Supervised Learning and Classification

- Perceptrons
- Linear Regression via a Probabilistic Interpretation
- Logistic Regression

Linear Classification: Mushroom and Goats

	color	width	height	label
0	-0.311688	0.358501	0.936567	edible
1	-0.472327	0.817906	0.468387	poisonous

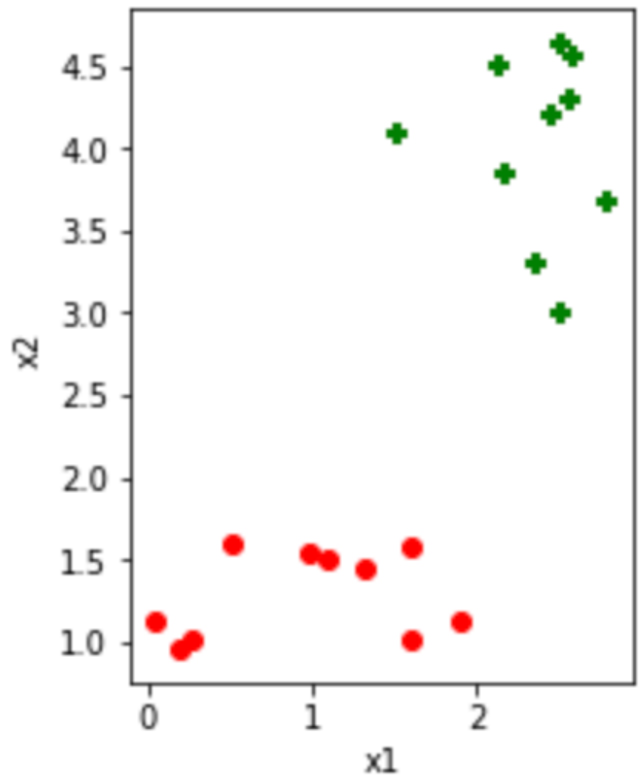
$$\mathbf{sign}(w_c * \mathbf{color} + w_w * \mathbf{width} + w_h * \mathbf{height})$$

$$\mathbf{sign}(0 * -0.472327 + 1 * 0.817906 - 1 * 0.468387) = \mathbf{sign}(0.349519) = +1$$

$$\mathbf{sign}(0 * -0.311688 + 1 * 0.358501 - 1 * 0.936567) = \mathbf{sign}(-0.578066) = -1$$

Linear Classification

	x1	x2	y
0	0.048589	1.120275	-1
1	0.200023	0.956716	-1
2	1.595538	1.023582	-1
3	1.315929	1.452371	-1
4	1.087080	1.513219	-1
5	0.512235	1.594651	-1
6	0.265039	1.008506	-1
7	1.606480	1.571889	-1
8	0.977585	1.550227	-1
9	1.908708	1.121259	-1
10	2.503476	3.002576	1



Classification

Given a training set $\{(x^{(i)}, y^{(i)})$ for $i = 1, \ldots, n\}$ let $y^{(i)} \in \{0, 1\}$.
Why not use regression, say least squares? A picture . . .

Loss Function for Classification: 0-1 Loss

L_{0-1}	$\hat{y} = -1$	$\hat{y} = 1$
$y = -1$	0	1
$y = 1$	1	0

Loss Function for Classification: 0-1 Loss

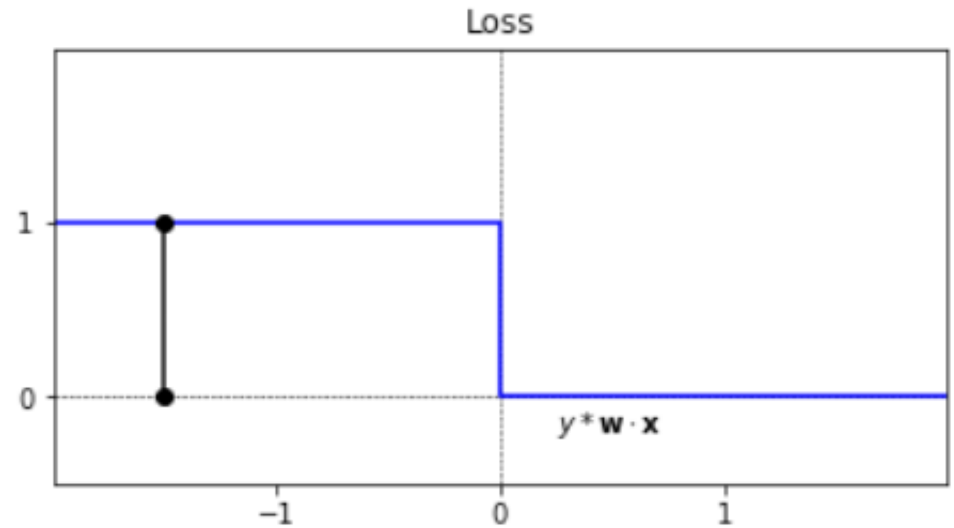
L_{0-1}	$\hat{y} = -1$	$\hat{y} = 1$
$y = -1$	0	1
$y = 1$	1	0

$$L_{0-1}(y, \mathbf{w} \cdot \mathbf{x}) = \begin{cases} 0 & \text{if } y * \mathbf{w} \cdot \mathbf{x} > 0 \\ 1 & \text{otherwise} \end{cases}$$

Loss Function for Classification: 0-1 Loss

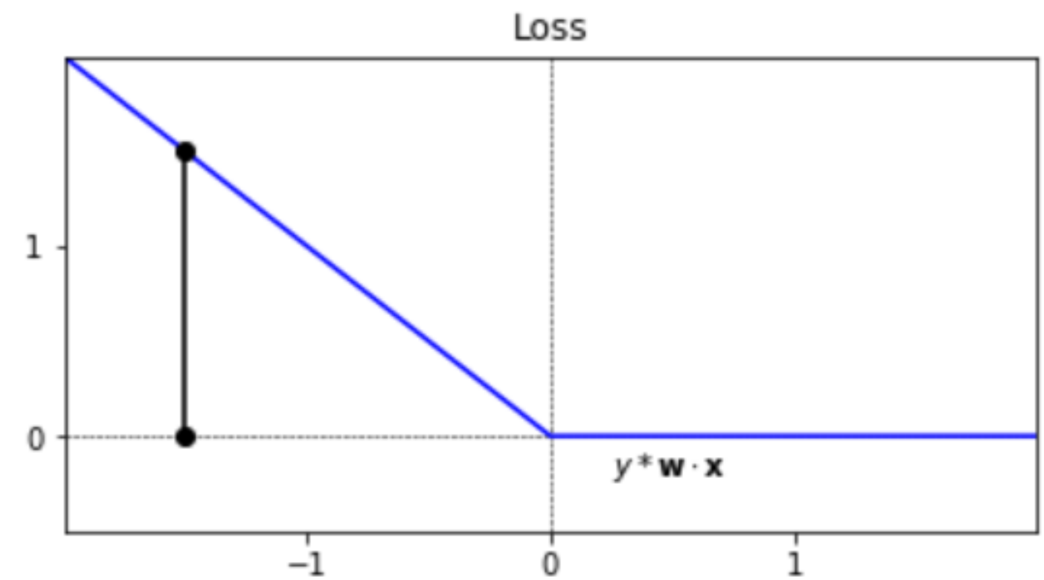
L_{0-1}	$\hat{y} = -1$	$\hat{y} = 1$
$y = -1$	0	1
$y = 1$	1	0

$$L_{0-1}(y, \mathbf{w} \cdot \mathbf{x}) = \begin{cases} 0 & \text{if } y * \mathbf{w} \cdot \mathbf{x} > 0 \\ 1 & \text{otherwise} \end{cases}$$



Perceptron Loss

$$L_P(y, \mathbf{w}\cdot\mathbf{x}) = \begin{cases} 0 & \text{if } y * \mathbf{w}\cdot\mathbf{x} > 0 \\ -y * \mathbf{w}\cdot\mathbf{x} & \text{otherwise} \end{cases}$$

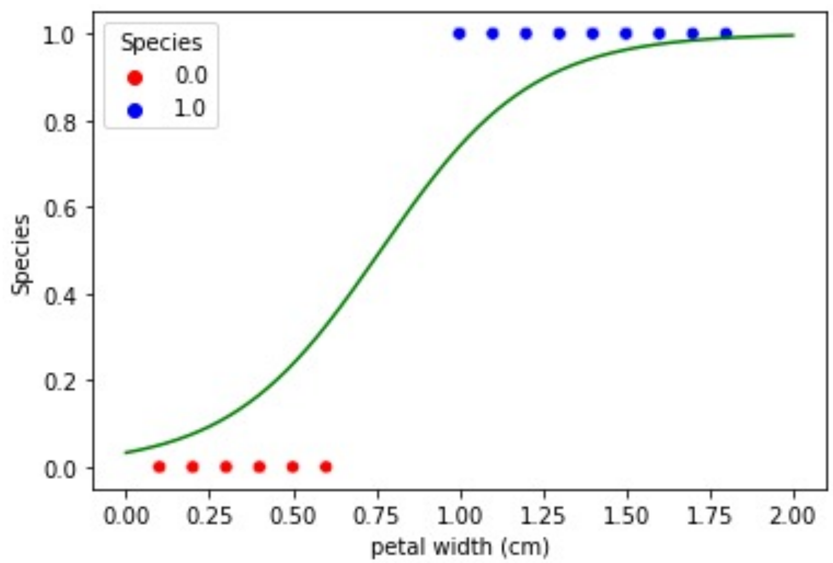


```python
def perceptron(df, label = 'y', epochs = 100, bias = True):

    if bias:
        df = df.copy()
        df.insert(0, '_x0_', 1)

    w = np.zeros(len(df.columns) - 1)
    features = [column for column in df.columns if column != label]

    for _ in range(epochs):
        errors = 0
        for _, row in df.iterrows():
            x = row[features]
            y = row[label]
            if y * np.dot(w, x) <= 0:
                w = w + y * x
                errors += 1
            yield w.copy()
        if errors == 0:
            break
```



Graph of Iris Dataset with logistic regression

Logistic Regression: Link Functions

Given a training set $\{(x^{(i)}, y^{(i)})$ for $i = 1, \ldots, n\}$ let $y^{(i)} \in \{0, 1\}$.
Want $h_\theta(x) \in [0, 1]$. Let's pick a smooth function:

$$h_\theta(x) = g(\theta^T x)$$

Here, g is a link function. There are *many*...

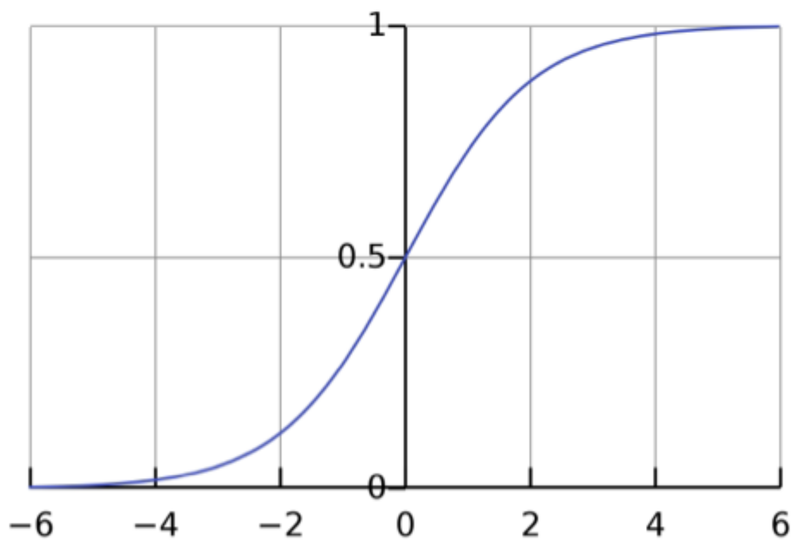
Logistic Regression: Link Functions

Given a training set $\{(x^{(i)}, y^{(i)})$ for $i = 1, \ldots, n\}$ let $y^{(i)} \in \{0, 1\}$.
Want $h_\theta(x) \in [0, 1]$. Let's pick a smooth function:

$$h_\theta(x) = g(\theta^T x)$$

Here, g is a link function. There are *many*... but we'll pick one!

$$g(z) = \frac{1}{1 + e^{-z}}.$$



Logistic Regression: Link Functions

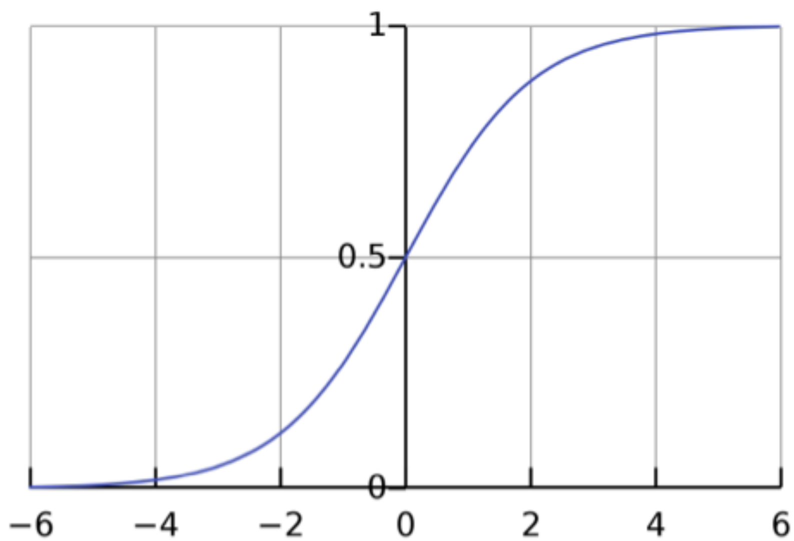
Given a training set $\{(x^{(i)}, y^{(i)}) \text{ for } i = 1, \ldots, n\}$ let $y^{(i)} \in \{0, 1\}$.
Want $h_\theta(x) \in [0, 1]$. Let's pick a smooth function:

$$h_\theta(x) = g(\theta^T x)$$

Here, g is a link function. There are *many*... but we'll pick one!

$$g(z) = \frac{1}{1 + e^{-z}}.$$

SIGMOID



How do we interpret $h_\theta(x)$?

$$P(y = 1 \mid x; \theta) = h_\theta(x)$$
$$P(y = 0 \mid x; \theta) = 1 - h_\theta(x)$$

Logistic Regression: Link Functions

Let's write the Likelihood function. Recall:

$$
\begin{aligned}
P(y = 1 \mid x; \theta) =& h_\theta(x) \\
P(y = 0 \mid x; \theta) =& 1 - h_\theta(x)
\end{aligned}
$$

Then,

$$L(\theta) = P(y \mid X; \theta) = \prod_{i=1}^{n} p(y^{(i)} \mid x^{(i)}; \theta)$$

Logistic Regression: Link Functions

Let's write the Likelihood function. Recall:

$$P(y = 1 \mid x; \theta) = h_\theta(x)$$
$$P(y = 0 \mid x; \theta) = 1 - h_\theta(x)$$

Then,

$$L(\theta) = P(y \mid X; \theta) = \prod_{i=1}^{n} p(y^{(i)} \mid x^{(i)}; \theta)$$

How do we go to a cost function from P (y | X; θ) ?

We need to go back to Maximum Likelihood Estimation that we saw before at the beginning of this lecture.

Logistic Regression: Link Functions

Let's write the Likelihood function. Recall:

$$\begin{aligned} P(y = 1 \mid x; \theta) =& h_\theta(x) \\ P(y = 0 \mid x; \theta) =& 1 - h_\theta(x) \end{aligned}$$

Then,

$$\begin{aligned} L(\theta) =& P(y \mid X; \theta) = \prod_{i=1}^{n} p(y^{(i)} \mid x^{(i)}; \theta) \\ =& \prod_{i=1}^{n} h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \quad \text{exponents encode ``if-then''} \end{aligned}$$

Logistic Regression: Link Functions

Let's write the Likelihood function. Recall:

$$\begin{aligned} P(y = 1 \mid x; \theta) =& h_\theta(x) \\ P(y = 0 \mid x; \theta) =& 1 - h_\theta(x) \end{aligned}$$

Then,

$$\begin{aligned} L(\theta) =& P(y \mid X; \theta) = \prod_{i=1}^{n} p(y^{(i)} \mid x^{(i)}; \theta) \\ =& \prod_{i=1}^{n} h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \quad \text{exponents encode ``if-then''} \end{aligned}$$

Taking logs to compute the log likelihood $\ell(\theta)$ we have:

$$\ell(\theta) = \log L(\theta) = \sum_{i=1}^{n} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

Now to solve it. . .

$$\ell(\theta) = \log L(\theta) = \sum_{i=1}^{n} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

We maximize for θ but we already saw how to do this! Just compute derivative, run (S)GD and you're done with it!

Takeaway: This is *another* example of the max likelihood method: we setup the likelihood, take logs, and compute derivatives.

Time Permitting: There is magic in the derivative. . .

Even more, the batch update can be written in a *remarkably familiar* form:

$$\theta^{(t+1)} = \theta^{(t)} + \sum_{j \in B} (y^{(j)} - h_\theta(x^{(j)}))x^{(j)}.$$

We sketch why (you can check!) We drop superscripts to simplify notation and examine a single data point:

$$\begin{aligned} & y \log h_\theta(x) + (1-y)\log(1-h_\theta(x)) \\ = & -y\log(1+e^{-\theta^T x}) + (1-y)(-\theta^T x) - (1-y)\log(1+e^{-\theta^T x}) \\ = & -\log(1+e^{-\theta^T x}) - (1-y)(\theta^T x) \end{aligned}$$

We used $1 - h_\theta(x) = \frac{e^{-\theta^T x}}{1-e^{-\theta^T x}}$. We now compute the derivative of this expression wrt θ and get:

$$\frac{e^{-\theta^T x}}{1+e^{-\theta^T x}}x - (1-y)x = (y - h_\theta(x))x$$

Perceptron Learning Algorithm

- Modify link function to output either 0 or 1.
- Make g to be a threshold function
- Then use same $h_\theta(x) = g(\theta^T x)$ using this g
- Follow the same update rule for θ

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Summary of Introduction to Classification

- We used the principle of maximum likelihood (and a probabilistic model) to extend to classification.

Summary of Introduction to Classification

- We used the principle of maximum likelihood (and a probabilistic model) to extend to classification.
- We developed logistic regression from this principle.
  - Logistic regression is *widely* used today.

Summary of Introduction to Classification

- We used the principle of maximum likelihood (and a probabilistic model) to extend to classification.
- We developed logistic regression from this principle.
  - Logistic regression is *widely* used today.
- We noticed a familiar pattern: take derivatives of the likelihood, and the derivatives had this (hopefully) intuitive *"misprediction form"*

Optimization Method Summary

Method	Compute per Step	Number of Steps to convergence
SGD	$\theta(d)$	$\approx \epsilon^{-2}$
Minibatch SGD		
GD	$\theta(nd)$	$\approx \epsilon^{-1}$
Newton	$\Omega(nd^2)$	$\approx \log(1/\epsilon)$

- In classical stats, d is small (< 100), n is often small, and *exact parameters matter*
- In modern ML, d is huge (billions, trillions), n is huge (trillions), and parameters used *only* for prediction
  - These are approximate number of computing steps
  - Convergence happens when loss settles to within an error range around the final value.
  - Newton would be very fast, where SGD needs a lot of step, but individual steps are fast, makes up for it
- As a result, (minibatch) SGD is the *workhorse* of ML.

Classification Lecture Summary

- We saw the differences between classification and regression.
- We learned about a principle for probabilistic interpretation for linear regression and classification: **Maximum Likelihood**.
  - We used this to derive logistic regression.
  - The Maximum Likelihood principle will be used again next lecture (and in the future)