

CMSC 478  
Lecture 4  
KMA Solaiman

Supervised Learning:  
Logistic Regression

Some slides are slightly adapted from Chris Re, Stanford ML

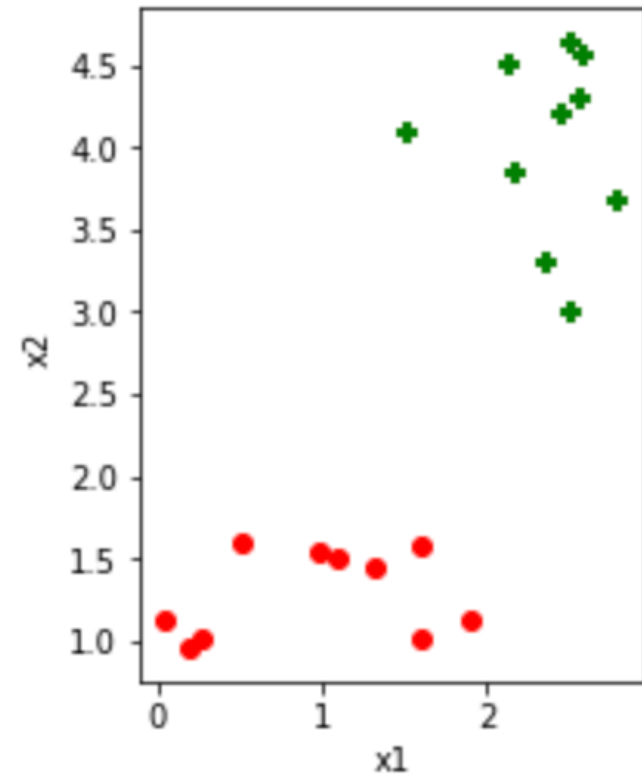
# Optimization Method Summary

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.

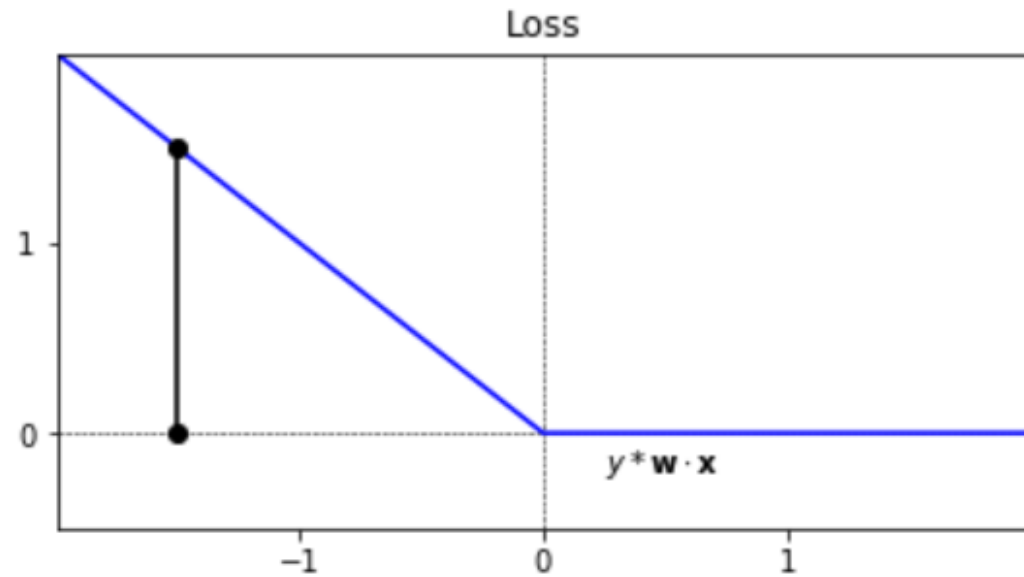
## Linear Classification

	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



# Perceptron Loss

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

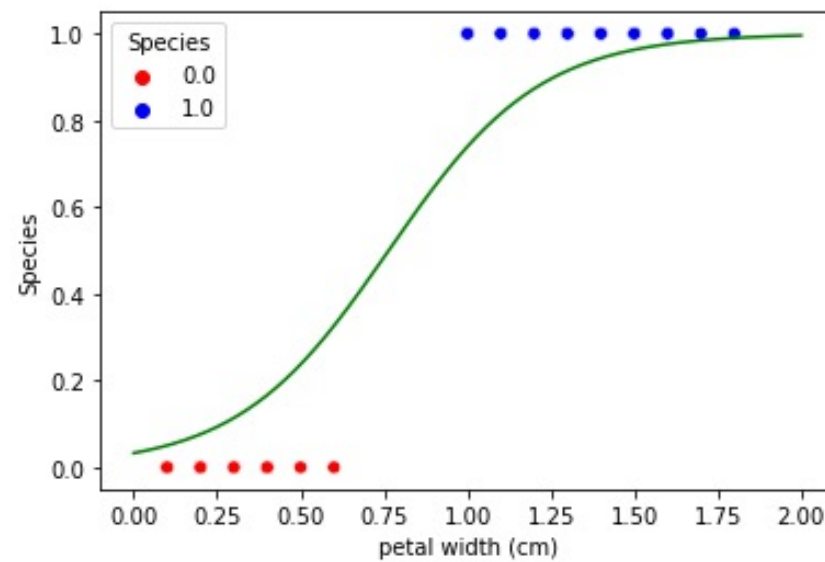


```
def perceptron(df, label = 'y', epochs = 100, bias = True):

    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  $\{(x^{(i)}, y^{(i)}) \text{ for } i = 1, \dots, n\}$  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*...

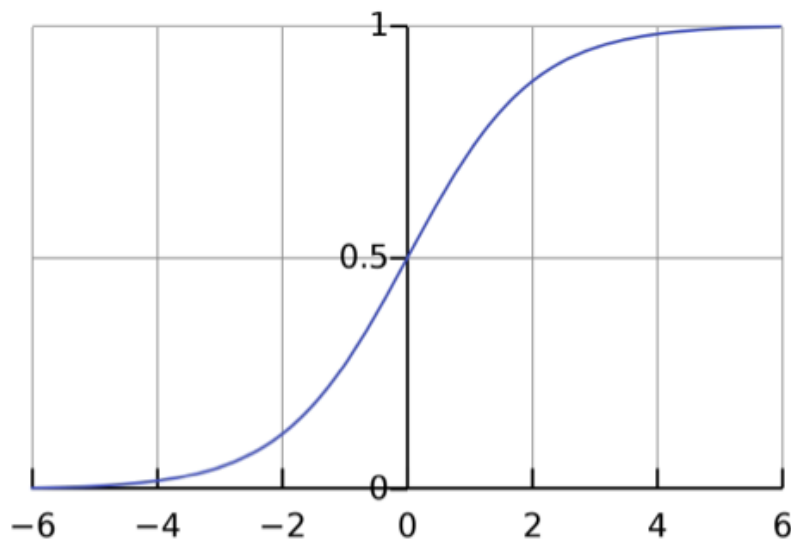
# Logistic Regression: Link Functions

Given a training set  $\{(x^{(i)}, y^{(i)}) \text{ for } i = 1, \dots, n\}$  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!

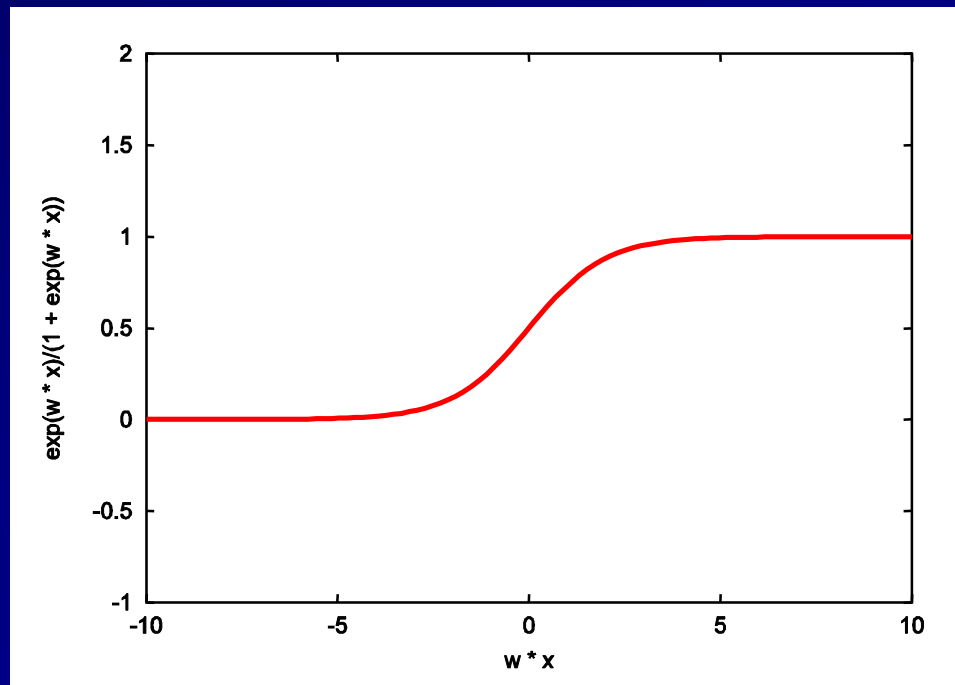
$$g(z) = \frac{1}{1 + e^{-z}}.$$





# Why the exp function?

- One reason: A linear function has a range from  $[-\infty, \infty]$  and we need to force it to be positive and sum to 1 in order to be a probability:



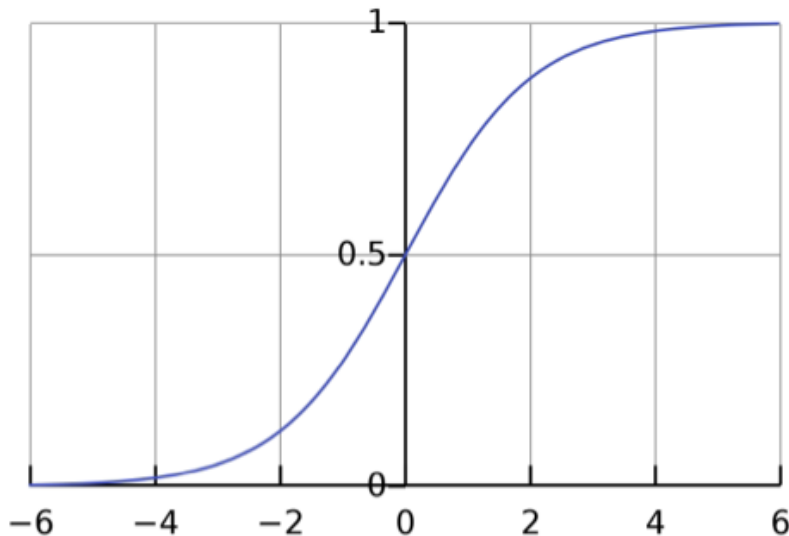
# Logistic Regression: Link Functions

Given a training set  $\{(x^{(i)}, y^{(i)}) \text{ for } i = 1, \dots, n\}$  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!

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



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:

$$P(y = 1 \mid x; \theta) = h_{\theta}(x)$$

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

Then,

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

# 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,

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

 Conditional Distribution  $P(y \mid X)$

# 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 something similar to a cost function from  $P(y \mid X; \theta)$  ?

- Maximum Likelihood Estimation (MLE)

# 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,

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

$$= \prod_{i=1}^n h_{\theta}(x^{(i)})^{y^{(i)}} (1 - h_{\theta}(x^{(i)}))^{1-y^{(i)}} \quad \text{exponents encode "if-then"}$$

# 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,

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

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  $\theta$  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  $\theta$  and get:

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

# Batch Gradient Ascent for Logistic Regression

**Given:** training examples  $(\mathbf{x}^i, y^i)$ ,  $i = 1 \dots N$

**Let**  $\theta = (0, 0, 0, 0, \dots, 0)$  be the initial weight vector.

**Repeat** until convergence

**Let**  $\nabla = (0, 0, \dots, 0)$  be the gradient vector.

**For**  $i = 1$  **to**  $N$  **do**

$$p^i = 1 / (1 + \exp[-\mathbf{w} \cdot \mathbf{x}^i])$$

$$\text{error}^i = y^i - p^i$$

**For**  $j = 1$  **to**  $d$  **do**

$$\nabla_j = \nabla_j + \text{error}^i \cdot x_j^i$$

$\theta := \theta + \alpha \cdot \nabla$  step in direction of increasing gradient

- An online gradient ascent algorithm can be constructed, of course
- Most statistical packages use a second-order (Newton-Raphson) algorithm for faster convergence. Each iteration of the second-order method can be viewed as a weighted least squares computation, so the algorithm is known as Iteratively-Reweighted Least Squares (IRLS)

# 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  $\theta$

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

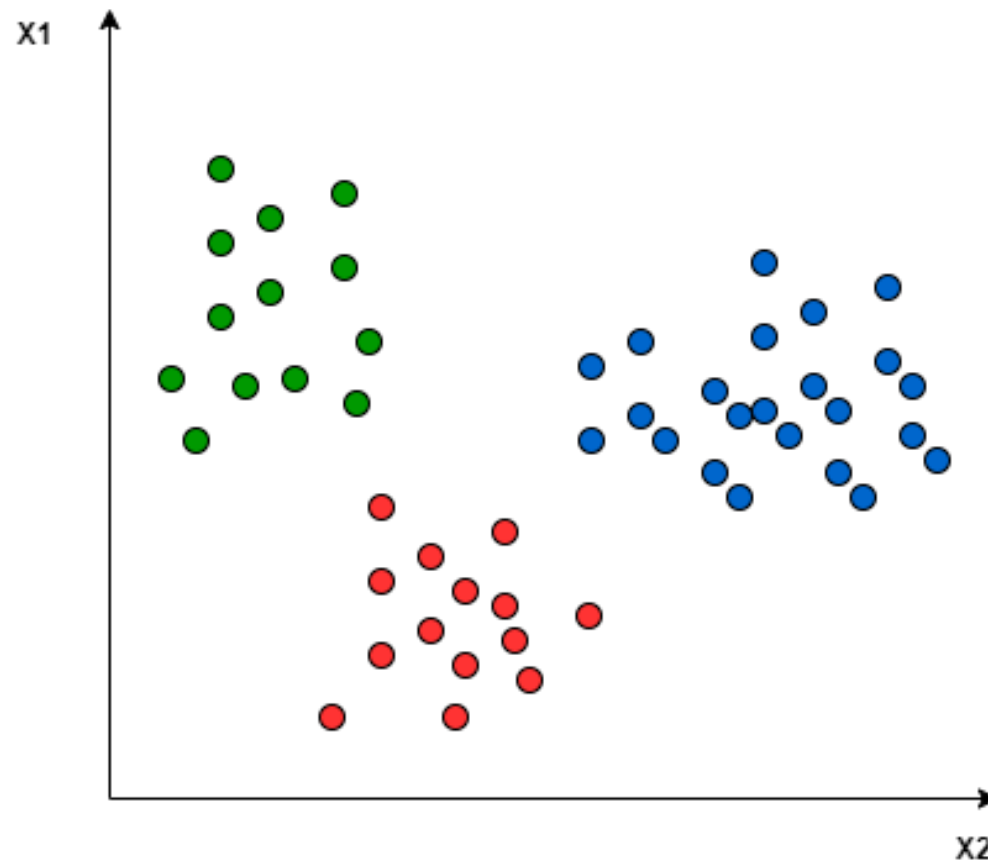
# Logistic Regression Implements a Linear Discriminant Function

- In the 2-class 0/1 loss function case, we should predict  $\hat{y} = 1$  if

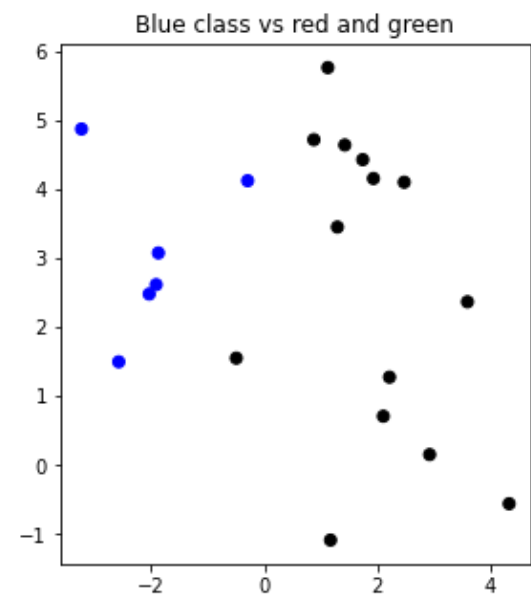
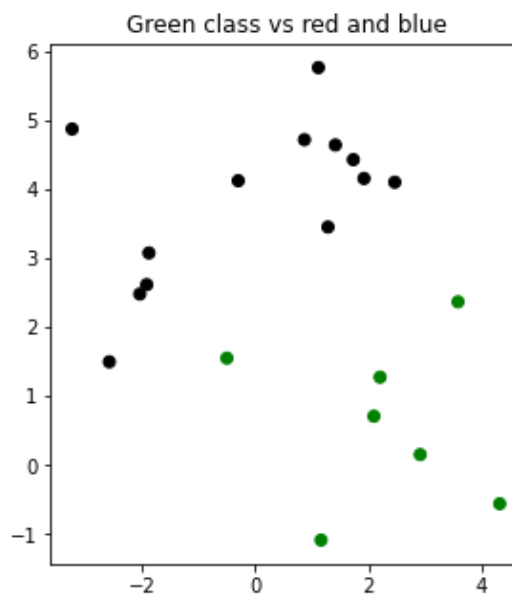
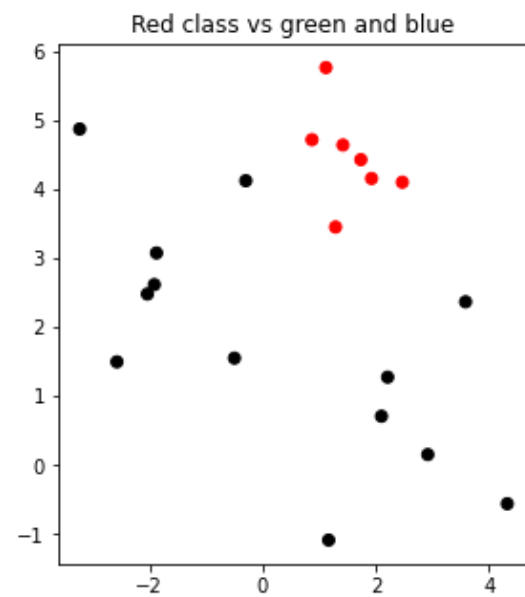
$$\begin{aligned} E_{y|\mathbf{x}}[L(0, y)] &> E_{y|\mathbf{x}}[L(1, y)] \\ \sum_y P(y|\mathbf{x})L(0, y) &> \sum_y P(y|\mathbf{x})L(1, y) \\ P(y = 0|\mathbf{x})L(0, 0) + P(y = 1|\mathbf{x})L(0, 1) &> P(y = 0|\mathbf{x})L(1, 0) + P(y = 1|\mathbf{x})L(1, 1) \\ P(y = 1|\mathbf{x}) &> P(y = 0|\mathbf{x}) \\ \frac{P(y = 1|\mathbf{x})}{P(y = 0|\mathbf{x})} &> 1 \quad \text{if } P(y = 0|\mathbf{x}) \neq 0 \\ \log \frac{P(y = 1|\mathbf{x})}{P(y = 0|\mathbf{x})} &> 0 \\ \mathbf{w} \cdot \mathbf{x} &> 0 \end{aligned}$$

- A similar derivation can be done for arbitrary  $L(0, 1)$  and  $L(1, 0)$ .

Extending LR to  $K > 2$  classes



# 1 vs All



A Quick and Dirty Intro to Multiclass Classification.  
This technique is *the daily workhorse of modern AI/ML*

# Multiclass

Suppose we want to choose among  $k$  discrete values, e.g.,  $\{\text{'Cat'}, \text{'Dog'}, \text{'Car'}, \text{'Bus'}\}$  so  $k = 4$ .

We encode with **one-hot** vectors i.e.  $y \in \{0, 1\}^k$  and  $\sum_{j=1}^k y_j = 1$ .

$$\begin{matrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \\ \text{'Cat'} & \text{'Dog'} & \text{'Car'} & \text{'Bus'} \end{matrix}$$

A prediction here is actually a *distribution* over the  $k$  classes. This leads to the SOFTMAX function described below (derivation in the notes!). That is our hypothesis is a vector of  $k$  values:

$$P(y = j|x; \bar{\theta}) = \frac{\exp(\theta_j^T x)}{\sum_{i=1}^k \exp(\theta_i^T x)}.$$

Here each  $\theta_j$  has the *same dimension* as  $x$ , i.e.,  $x, \theta_j \in R^{d+1}$  for  $j = 1, \dots, k$ .



## Extending Logistic Regression to $K > 2$ classes

- Choose class  $K$  to be the “reference class” and represent each of the other classes as a logistic function of the odds of class  $k$  versus class  $K$ :

$$\begin{aligned}\log \frac{P(y = 1|\mathbf{x})}{P(y = K|\mathbf{x})} &= \mathbf{w}_1 \cdot \mathbf{x} \\ \log \frac{P(y = 2|\mathbf{x})}{P(y = K|\mathbf{x})} &= \mathbf{w}_2 \cdot \mathbf{x} \\ &\vdots \\ \log \frac{P(y = K - 1|\mathbf{x})}{P(y = K|\mathbf{x})} &= \mathbf{w}_{K-1} \cdot \mathbf{x}\end{aligned}$$

- Gradient ascent can be applied to simultaneously train all of these weight vectors

$\mathbf{w}_k$

# Summary of Introduction to Classification

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

# Deriving a Learning Algorithm

- Since we are fitting a conditional probability distribution, we no longer seek to minimize the loss on the training data. Instead, we seek to find the probability distribution  $h$  that is most likely given the training data
- Let  $S$  be the training sample. Our goal is to find  $h$  to maximize  $P(h | S)$ :

$$\begin{aligned}\operatorname{argmax}_h P(h|S) &= \operatorname{argmax}_h \frac{P(S|h)P(h)}{P(S)} && \text{by Bayes' Rule} \\ &= \operatorname{argmax}_h P(S|h)P(h) && \text{because } P(S) \text{ doesn't depend on } h \\ &= \operatorname{argmax}_h P(S|h) && \text{if we assume } P(h) = \text{uniform} \\ &= \operatorname{argmax}_h \log P(S|h) && \text{because log is monotonic}\end{aligned}$$

The distribution  $P(S|h)$  is called the likelihood function. The log likelihood is frequently used as the objective function for learning. It is often written as  $\ell(\mathbf{w})$ .

The  $h$  that maximizes the likelihood on the training data is called the maximum likelihood estimator (MLE)

# 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*”

# Computing the Likelihood

- In our framework, we assume that each training example  $(\mathbf{x}_i, y_i)$  is drawn from the same (but unknown) probability distribution  $P(\mathbf{x}, y)$ . This means that the log likelihood of  $S$  is the sum of the log likelihoods of the individual training examples:

$$\begin{aligned}\log P(S|h) &= \log \prod_i P(\mathbf{x}^i, y^i|h) \\ &= \sum_i \log P(\mathbf{x}^i, y^i|h)\end{aligned}$$

# Computing the Likelihood (2)

- Recall that *any* joint distribution  $P(a,b)$  can be factored as  $P(a|b) P(b)$ . Hence, we can write

$$\begin{aligned}\operatorname{argmax}_h \log P(S|h) &= \operatorname{argmax}_h \sum_i \log P(\mathbf{x}^i, y^i|h) \\ &= \operatorname{argmax}_h \sum_i \log P(y^i|\mathbf{x}^i, h) P(\mathbf{x}^i|h)\end{aligned}$$

- In our case,  $P(\mathbf{x} | h) = P(\mathbf{x})$ , because it does not depend on  $h$ , so

$$\begin{aligned}\operatorname{argmax}_h \log P(S|h) &= \operatorname{argmax}_h \sum_i \log P(y^i|\mathbf{x}^i, h) P(\mathbf{x}^i|h) \\ &= \operatorname{argmax}_h \sum_i \log P(y^i|\mathbf{x}^i, h)\end{aligned}$$

# 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)