

CSE/STAT 416

Logistic Regression

Pemi Nguyen

Paul G. Allen School of Computer Science & Engineering
University of Washington

April 13, 2022



Logistics

Next two weeks:

Week 4: Other ML models for classification

Week 5: Fairness and societal impact of machine learning

- No pre-lecture videos
- There will still be checkpoints posted on EdStem

Homework 2 due this Friday

Tuesday of Week 6: Midterm exam (takehome)

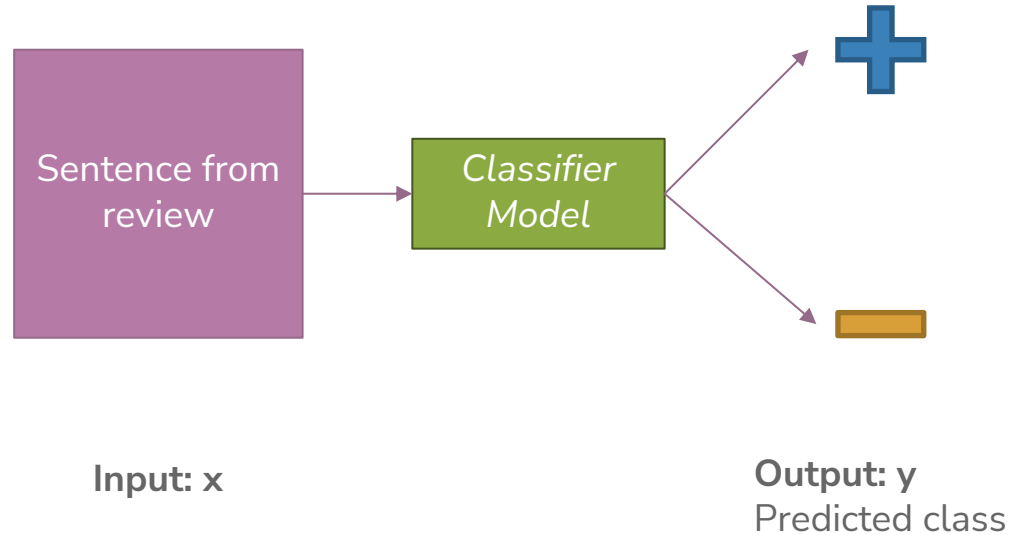
- Information on EdStem

Please try to ask questions on EdStem as much as you can.
I'll try my best to respond quickly



Sentiment Classifier

In our example, we want to classify a restaurant review as positive or negative.



Implementation 2: Linear Classifier

Idea: Use labelled training data to learn a weight for each word.
Use weights to score a sentence.

See last slide for example weights and scoring.

Linear Classifier

Input x : Sentence from review

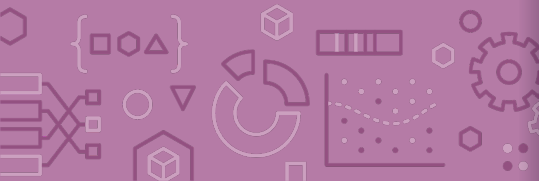
Compute $Score(x)$

If $Score(x) > 0$:

- $\hat{y} = +1$

Else:

- $\hat{y} = -1$

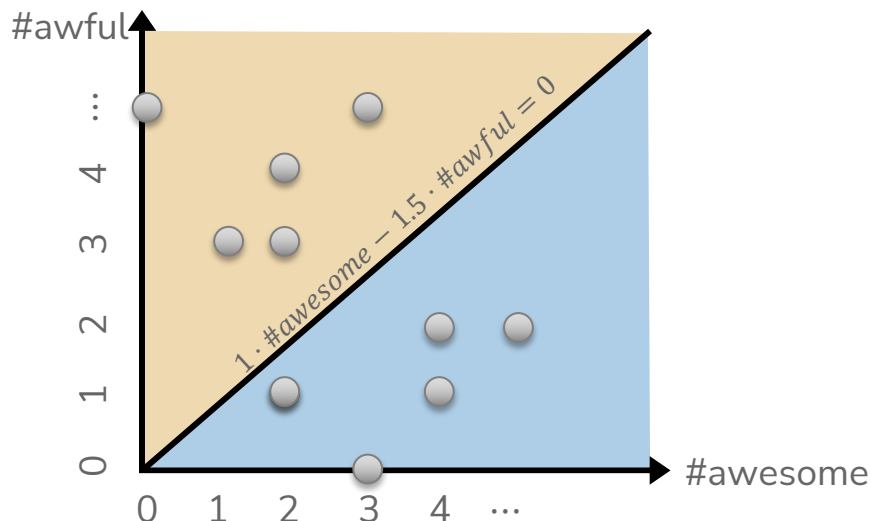


Decision Boundary

Consider if only two words had non-zero coefficients

Word	Coefficient	Weight
	w_0	0.0
awesome	w_1	1.0
awful	w_2	-1.5

$$\hat{s} = 1 \cdot \#awesome - 1.5 \cdot \#awful$$



Learning \hat{w}

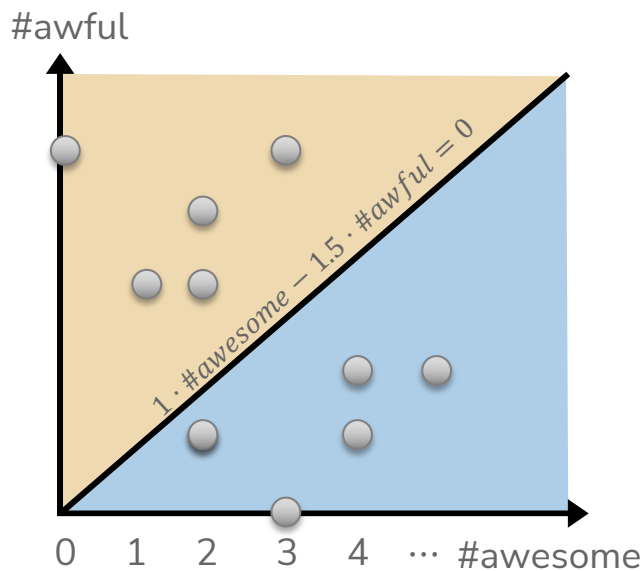
Can we use MSE for classification task?

One idea is to just model the processing of finding \hat{w} based on what we discussed in linear regression using MSE

$$\hat{w} = \underset{w}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n \mathbb{I}\{y_i \neq \hat{y}_i\}^2$$

Will this work?

Assume $h_1(x) = \#awesome$ so w_1 is its coefficient and w_2 is fixed.



How to measure error for classification

The MSE loss function doesn't work because of different reasons:

The outputs are discrete values with no ordinal nature, so we need a different way to frame how close a prediction is to a certain correct category

The MSE loss function for classification task is not continuous, differentiable or convex, so we can't use optimization algorithm like Gradient Descent to find an optimal set of weights

Note: Convexity is an important concept in Machine Learning. By minimizing error, we want to find where that global minimum is, and that's ideal in a convex function.

Let's frame this problem in term of probabilities instead.



Probabilities

Assume that there is some randomness in the world, and instead will try to model the probability of a positive/negative label.

Examples:

“The sushi & everything else were awesome!”

Definite positive (+1)

$$P(y = +1 \mid x = \text{“The sushi & everything else were awesome!”}) = 0.99$$

“The sushi was alright, the service was OK”

Not as sure

$$P(y = -1 \mid x = \text{“The sushi alright, the service was okay!”}) = 0.5$$

Use probability as the measurement of certainty

$$P(y|x)$$

Probability Classifier

Idea: Estimate probabilities $\hat{P}(y|x)$ and use those for prediction

Probability Classifier

Input x : Sentence from review

Estimate class probability $\hat{P}(y = +1|x)$

If $\hat{P}(y = +1|x) > 0.5$:

- $\hat{y} = +1$

Else:

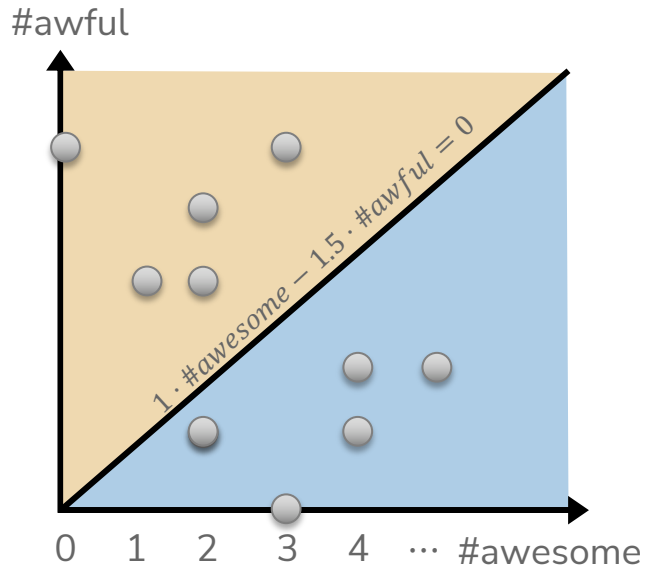
- $\hat{y} = -1$

Notes:

Estimating the probability improves **interpretability**

Score Probabilities?

Idea: Let's try to relate the value of $Score(x)$ to $\hat{P}(y = +1|x)$

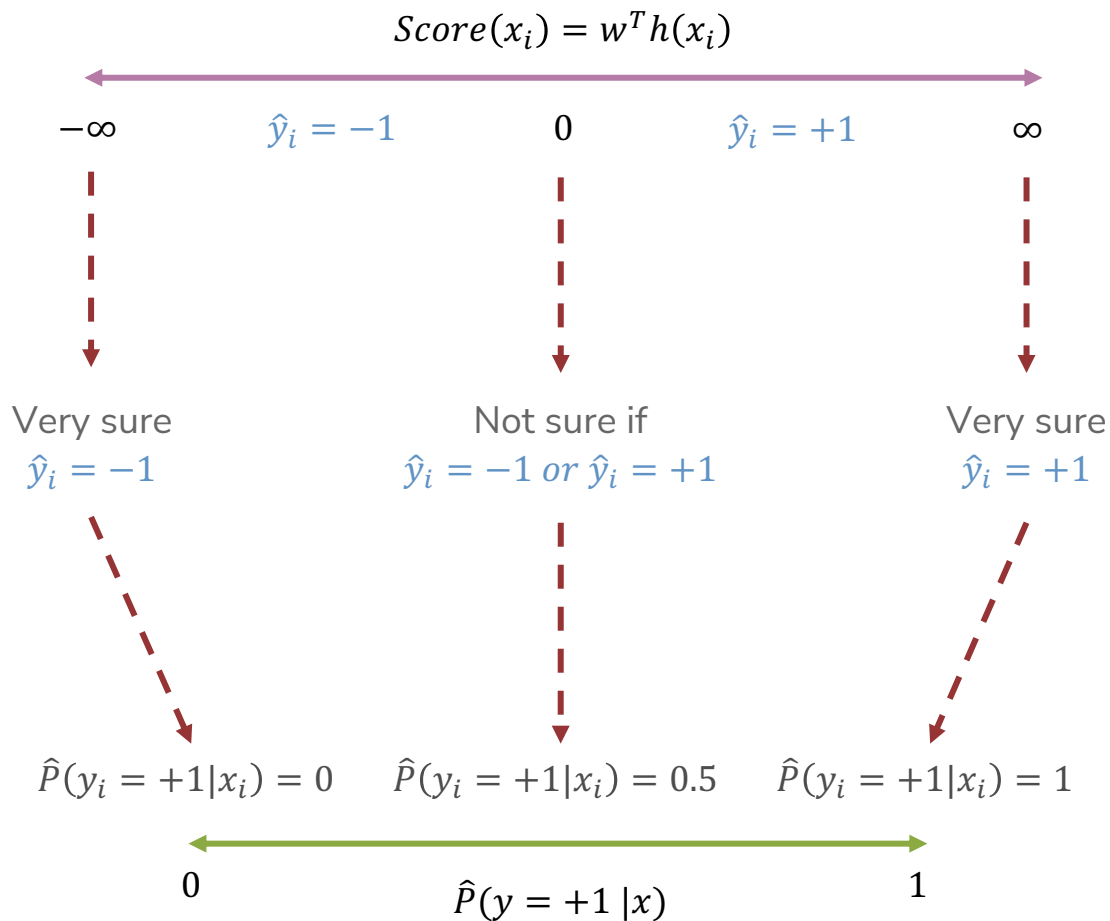


What if $Score(x)$ is positive?

What if $Score(x)$ is negative?

What if $Score(x)$ is 0?

Interpreting Score

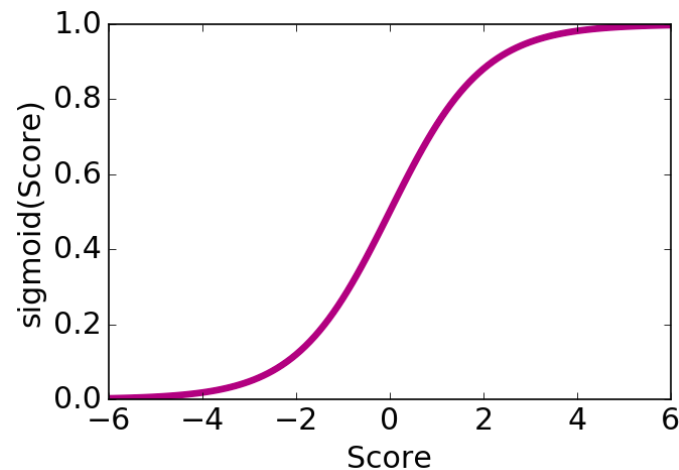


Logistic Function

Use a function that takes numbers arbitrarily large/small and maps them between 0 and 1.

$$\text{sigmoid}(\text{Score}(x)) = \frac{1}{1 + e^{-\text{Score}(x)}}$$

$\text{Score}(x)$	$\text{sigmoid}(\text{Score}(x))$
$-\infty$	0
-2	0.12
0	0.5
2	0.88
∞	1



Logistic Regression Model

$$P(y_i = +1|x_i, w) = \text{sigmoid}(\text{Score}(x_i)) = \frac{1}{1 + e^{-w^T h(x_i)}}$$

Logistic Regression Classifier

Input x : Sentence from review

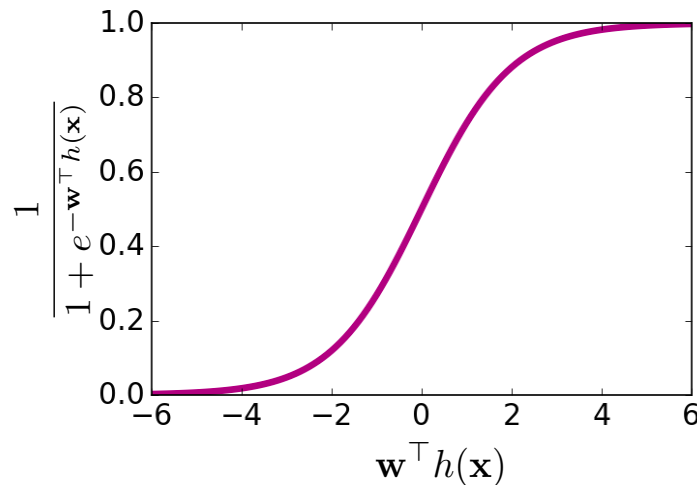
Estimate class probability $\hat{P}(y = +1|x, \hat{w}) = \text{sigmoid}(\hat{w}^T h(x_i))$

If $\hat{P}(y = +1|x, \hat{w}) > 0.5$:

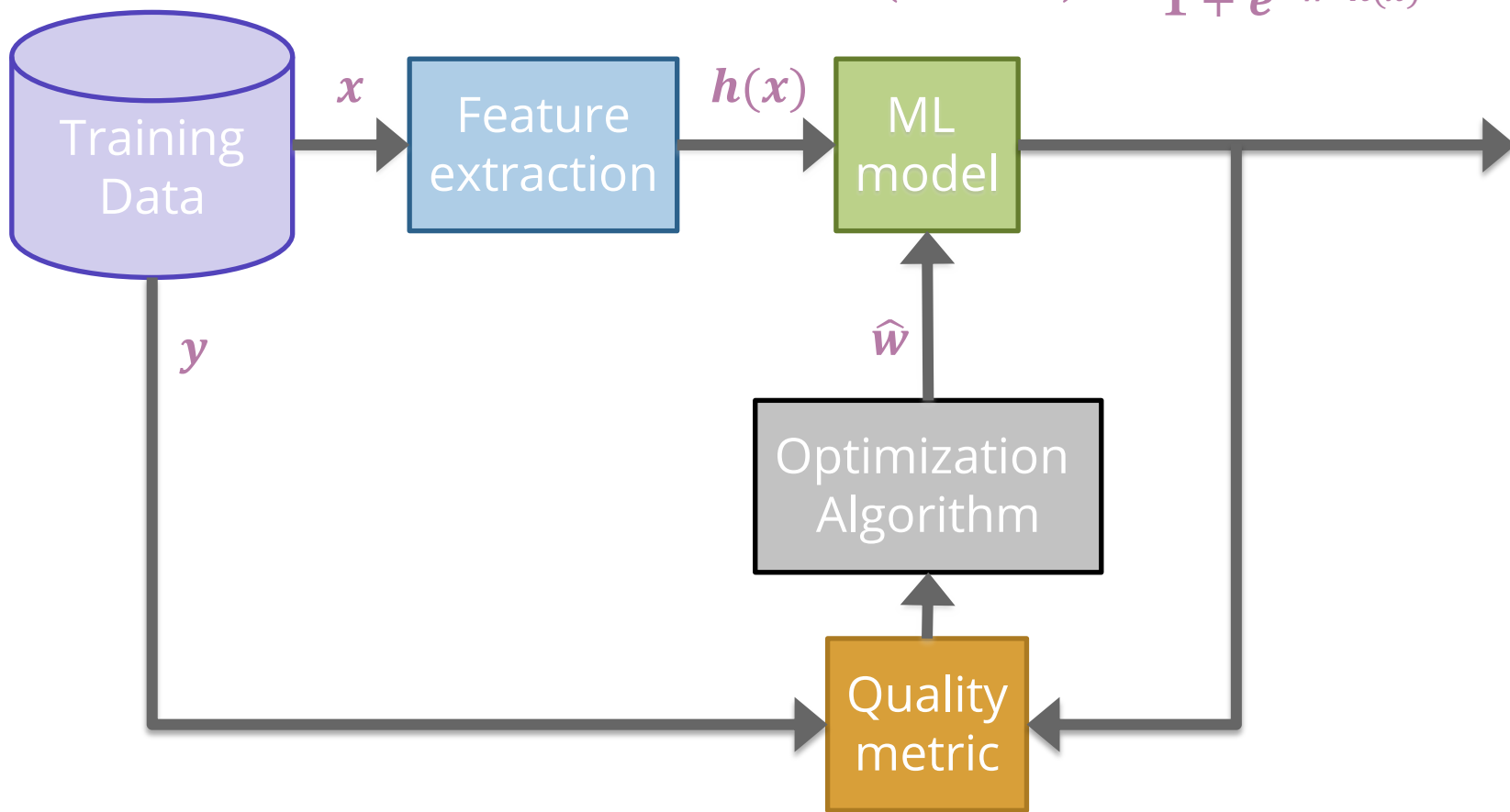
- $\hat{y} = +1$

Else:

- $\hat{y} = -1$



$$\hat{P}(y = +1|x, \hat{w}) = \text{sigmoid}(\hat{w}^T h(x)) = \frac{1}{1 + e^{-\hat{w}^T h(x)}}$$



Class Session

Quality Metric = Likelihood

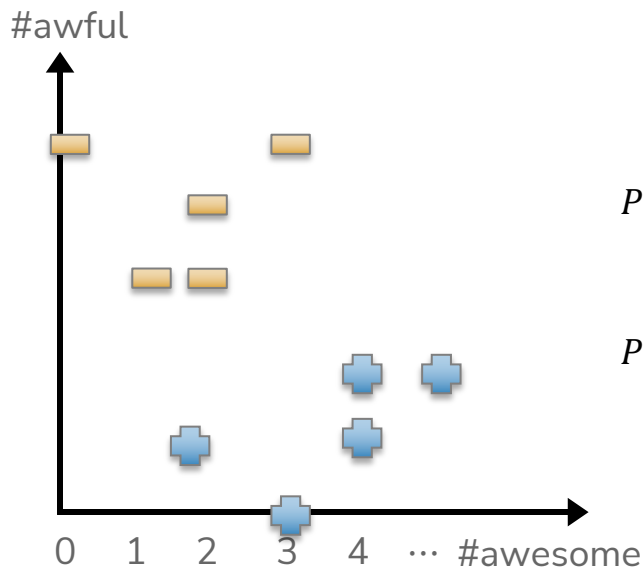
Want to compute the probability of seeing our dataset for every possible setting for w . Find w that makes data most likely!

Data Point	$h_1(x)$	$h_2(x)$	y	Choose w to maximize
$x^{(1)}, y^{(1)}$	2	1	+1	$P(y^{(1)} = +1 x^{(1)}, w)$
$x^{(2)}, y^{(2)}$	0	2	-1	$P(y^{(2)} = -1 x^{(2)}, w)$
$x^{(3)}, y^{(3)}$	3	3	-1	$P(y^{(3)} = -1 x^{(3)}, w)$
$x^{(4)}, y^{(4)}$	4	1	+1	$P(y^{(4)} = +1 x^{(4)}, w)$

Learn \hat{w}

Now that we have our new model, we will talk about how to choose \hat{w} to be the “best fit”.

The choice of w affects how likely seeing our dataset is



$$\ell(w) = \prod_i^n P(y^{(i)}|x^{(i)}, w)$$

$$P(y^{(i)} = +1|x^{(i)}, w) = \frac{1}{1 + e^{-w^T h(x^{(i)})}}$$

$$P(y^{(i)} = -1|x^{(i)}, w) = \frac{e^{-w^T h(x^{(i)})}}{1 + e^{-w^T h(x^{(i)})}}$$

Maximum Likelihood Estimate (MLE)

Find the w that maximizes the likelihood

$$\hat{w} = \operatorname{argmax}_w \ell(w) = \operatorname{argmax}_w \frac{1}{n} \prod_{i=1}^n P(y^{(i)} | x^{(i)}, w)$$

$$= \operatorname{argmax}_w \frac{1}{n} \sum_{i=1}^n \log(P(y^{(i)} | x^{(i)}, w))$$

(taking the log to turn product into sum of logs)

$$= \operatorname{argmin}_w \frac{1}{n} \sum_{i=1}^n -\log(P(y^{(i)} | x^{(i)}, w))$$

(Negative Log-Likelihood Loss)



Optional:
Further
expansion of
the logistic
regression loss

Two ways to represent the loss: Using the linear score function:

$$f(x^{(i)}) = w^T h(x^{(i)})$$

When the labels $y^{(i)} \in \{-1, 1\}$

$$\begin{aligned}\hat{w} &= \operatorname{argmax}_w \frac{1}{n} \left[\sum_{i=1: y^{(i)}=+1}^n \ln \left(\frac{1}{1 + e^{-f(x^{(i)})}} \right) + \sum_{i=1: y^{(i)}=-1}^n \ln \left(\frac{e^{-f(x^{(i)})}}{1 + e^{-f(x^{(i)})}} \right) \right] \\ &= \operatorname{argmin}_w \frac{1}{n} \left[\sum_{i=1: y^{(i)}=+1}^n \ln \left(1 + e^{-f(x^{(i)})} \right) + \sum_{i=1: y^{(i)}=-1}^n -\ln \left(\frac{1}{1 + e^{f(x^{(i)})}} \right) \right] \\ &= \operatorname{argmin}_w \frac{1}{n} \left[\sum_{i=1: y^{(i)}=+1}^n \ln \left(1 + e^{-(1)f(x^{(i)})} \right) + \sum_{i=1: y^{(i)}=-1}^n \ln \left(1 + e^{-(-1)f(x^{(i)})} \right) \right] \\ &= \operatorname{argmin}_w \frac{1}{n} \sum_{i=1}^n \log \left(1 + e^{-y^{(i)} f(x^{(i)})} \right)\end{aligned}$$

When the labels $y^{(i)} \in \{0, 1\}$ (cross-entropy loss) (more commonly used)

$$= \operatorname{argmin}_w -\frac{1}{n} \sum_{i=1}^n y^{(i)} \log \left[\sigma \left(f(x^{(i)}) \right) \right] + (1 - y^{(i)}) \log \left[1 - \sigma \left(f(x^{(i)}) \right) \right]$$

Likelihood vs Error

In understanding how to measure error for the classification problem, we want to understand how close a prediction is to the correct class, which means we want to assign a high probability for a correct prediction, and low probability for an incorrect prediction

Likelihood and error are the inverse of each other:

Maximizing likelihood = Minimizing Error

Likelihood function of the Logistic Regression problem

$$\prod_{i=1}^n P(y^{(i)} | x^{(i)}, w)$$

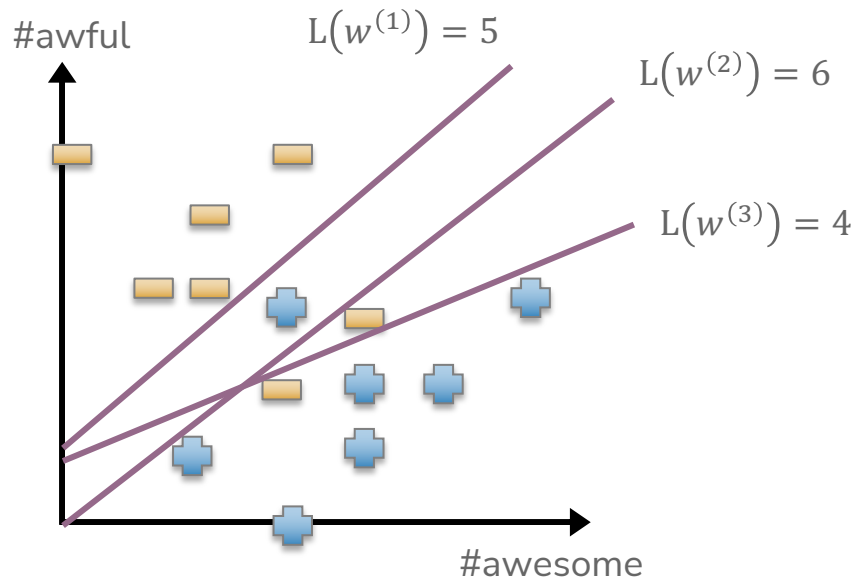
Negative log-likelihood loss function of the Logistic Regression problem:

$$L(w) = \sum_{i=1}^n -\log(P(y^{(i)} | x^{(i)}, w))$$

Think 

1 min

Which setting of w should we use?



pollev.com/cs416

1:00

What about multi-class classification?

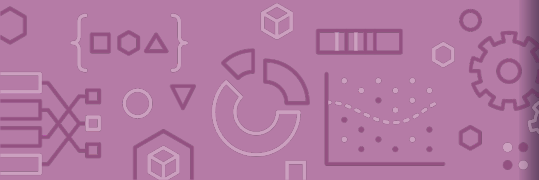
We just talked about using sigmoid to turn the score of an input into a probability, then use Negative Log-Likelihood to measure the loss of binary classification.

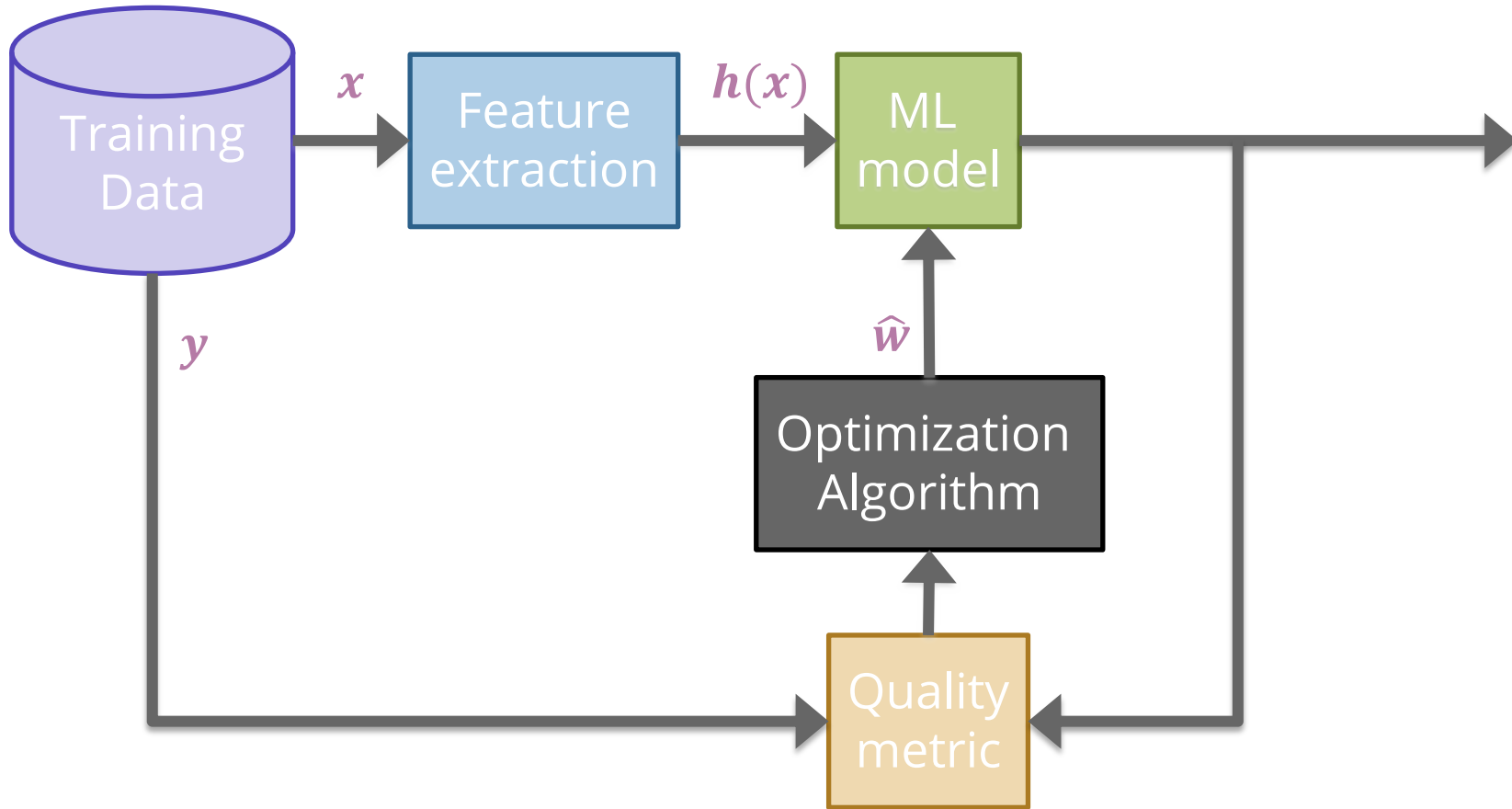
For multi-class classification (at least 3 classes), we will use **softmax** instead of **sigmoid**. (the details won't be covered in this class). We'll still use Negative Log-Likelihood as well. This combination is called **cross-entropy loss** (we'll revisit this in the Deep Learning week).





Brain Break

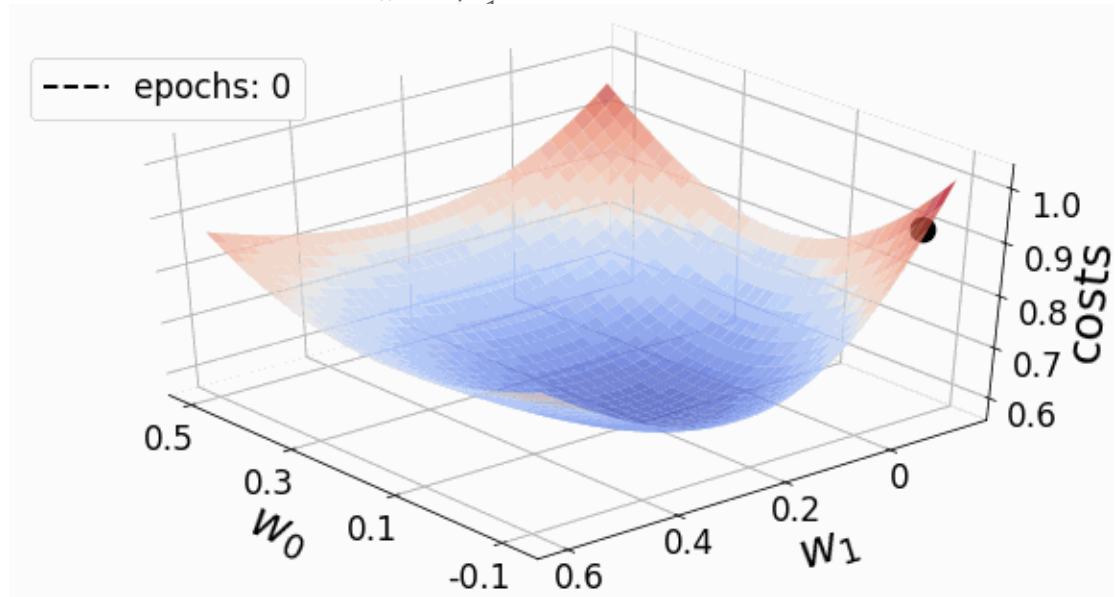




Finding optimal weights

- Linear Regression has a closed form solution
- However, Logistic Regression does not. We have to use an iterative method like **gradient descent**!

$$\hat{w} = \operatorname{argmin}_w \sum_{i=1}^n -\log(P(y^{(i)}|x^{(i)}, w))$$



Gradient Descent

Gradient Descent algorithm

Start at some (random) weights w

While we haven't converged:

$$w \leftarrow w - \alpha \nabla L(w)$$

- α : learning rate
- $\nabla L(w)$: the gradients of loss function L on a set of weights w

This is just describing going down the hill step by step.

α controls how big of steps we take, and picking it is crucial for how well the model you learn does!





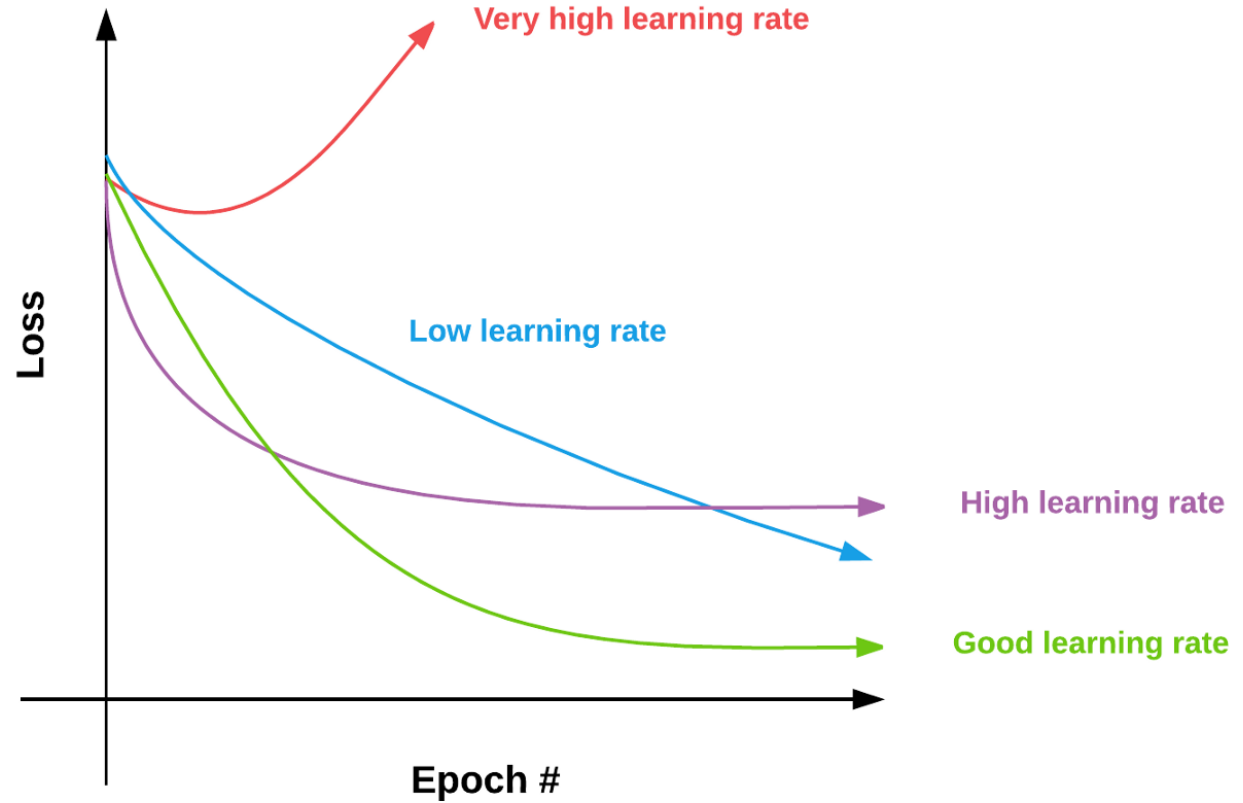
Think 

1 min

Note that α is a hyperparameter, and selecting the right value can lead to different results. What is the effect of very small and very large value of learning rate α during training?

pollev.com/cs416

Effects of different values of learning rates



Effects of different values of learning rates

Too large learning rate: the gradient might overshoot from the minimum and you will end up increasing the loss

Too small learning rate: take a long time for the loss to converge

Unfortunately, you have to do a lot of trial and error 😞

Try several values (generally exponentially spaced)

Find one that is too small and one that is too large to narrow search range. Try values in between!



Hyperparameter fine-tuning

We have introduced yet another hyperparameter that you have to choose, that will affect which predictor is ultimately learned.

If you want to tune both a Ridge penalty and a learning rate (step size for gradient descent), you will need to try all pairs of settings!

For example, suppose you wanted to try using a validation set to select the right settings out of:

- $\lambda \in [0.01, 0.1, 1, 10, 100]$
- $\in \left[0.001, 0.01, 0.1, 1, \frac{1}{t}, \frac{10}{t}\right]$

You will need to train 30 different models and evaluate each one! This is called **Grid Search**.

To decrease the number of combinations we have to choose, we can also use **Random Search**.

Some advanced issues with gradient descent

Advanced: Divergence with large step sizes tends to happen at the end, close to the optimal point. You can use a decreasing step size to avoid this

$$\alpha_t = \frac{\alpha_0}{t}$$

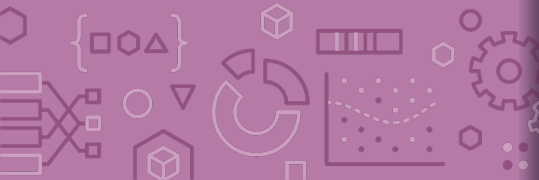
(t : number of iterations)

Advanced: For each iteration of gradient descent, we have to calculate the loss from each datapoint in order to calculate the gradient of an update. If you have a lot of training datapoints, let's say a million, calculating the gradient will be very computational expensive.

Instead, for each iteration, we can select a small number of random datapoints from the train set to calculate the gradients (instead of using the full train set) to decrease the computational time. This is called **Stochastic Gradient Descent**.



Brain Break



Overfitting - Classification

More Features

Like with regression, we can learn more complicated models by including more features or by including more complex features.

Instead of just using

$$h_1(x) = \#awesome$$

$$h_2(x) = \#awful$$

We could use

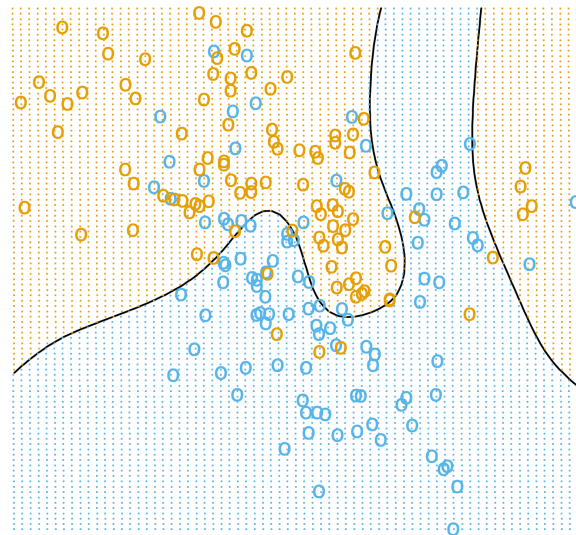
$$h_1(x) = \#awesome$$

$$h_2(x) = \#awful$$

$$h_3(x) = \#awesome^2$$

$$h_4(x) = \#awful^2$$

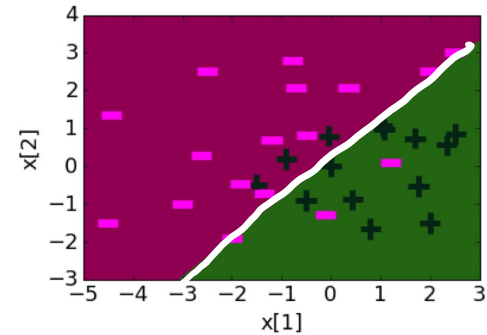
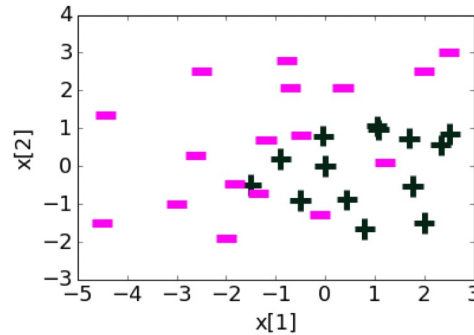
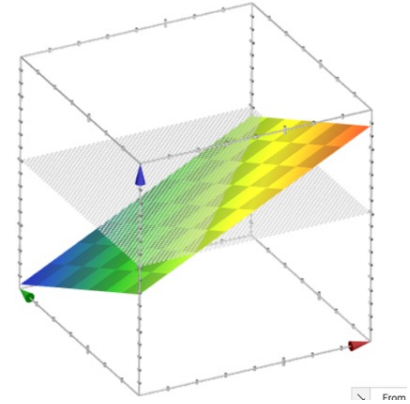
...



Decision Boundary

$$w^T h(x) = 0.23 + 1.12x[1] - 1.07x[2]$$

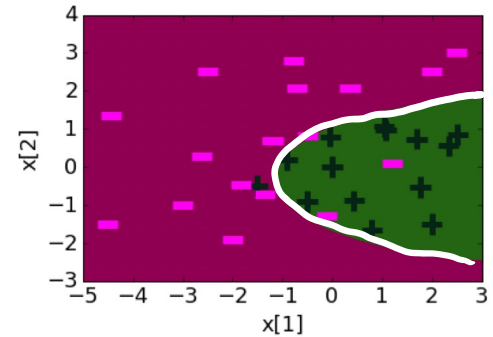
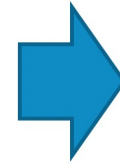
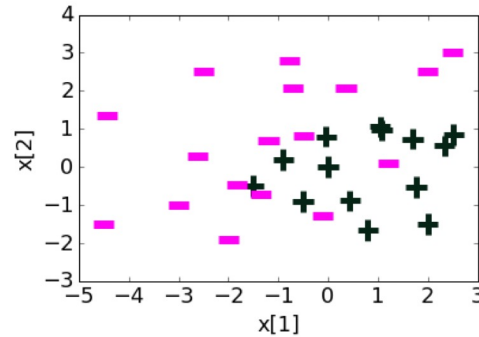
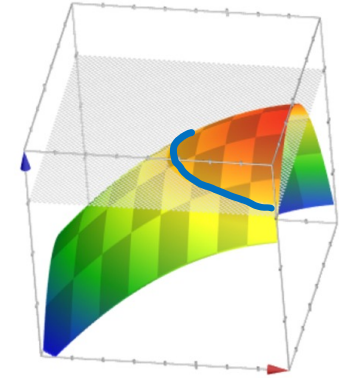
Feature	Value	Coefficient learned
$h_0(x)$	1	0.23
$h_1(x)$	$x[1]$	1.12
$h_2(x)$	$x[2]$	-1.07



Decision Boundary

$$w^T h(x) = 1.68 + 1.39x[1] - 0.59x[2] - 0.17x[1]^2 - 0.96x[2]^2$$

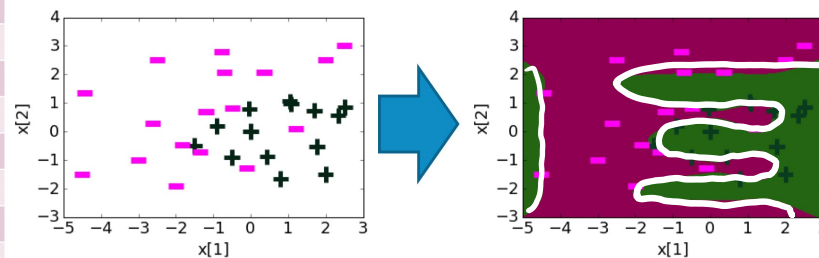
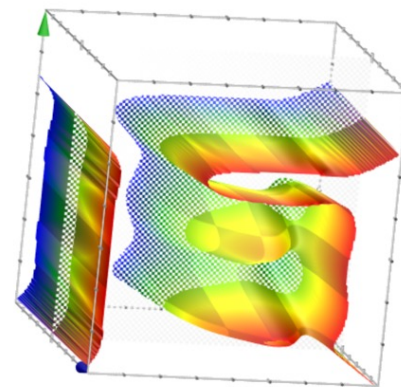
Feature	Value	Coefficient learned
$h_0(x)$	1	1.68
$h_1(x)$	$x[1]$	1.39
$h_2(x)$	$x[2]$	-0.59
$h_3(x)$	$(x[1])^2$	-0.17
$h_4(x)$	$(x[2])^2$	-0.96



Decision Boundary

$$w^T h(x) = \dots$$

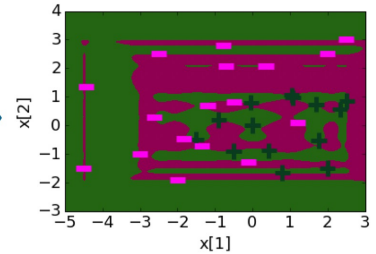
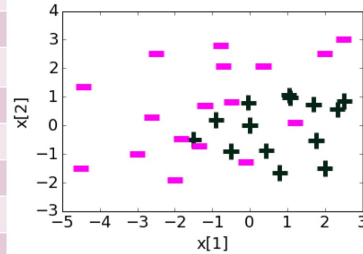
Feature	Value	Coefficient learned
$h_0(x)$	1	21.6
$h_1(x)$	$x[1]$	5.3
$h_2(x)$	$x[2]$	-42.7
$h_3(x)$	$(x[1])^2$	-15.9
$h_4(x)$	$(x[2])^2$	-48.6
$h_5(x)$	$(x[1])^3$	-11.0
$h_6(x)$	$(x[2])^3$	67.0
$h_7(x)$	$(x[1])^4$	1.5
$h_8(x)$	$(x[2])^4$	48.0
$h_9(x)$	$(x[1])^5$	4.4
$h_{10}(x)$	$(x[2])^5$	-14.2
$h_{11}(x)$	$(x[1])^6$	0.8
$h_{12}(x)$	$(x[2])^6$	-8.6



Decision Boundary

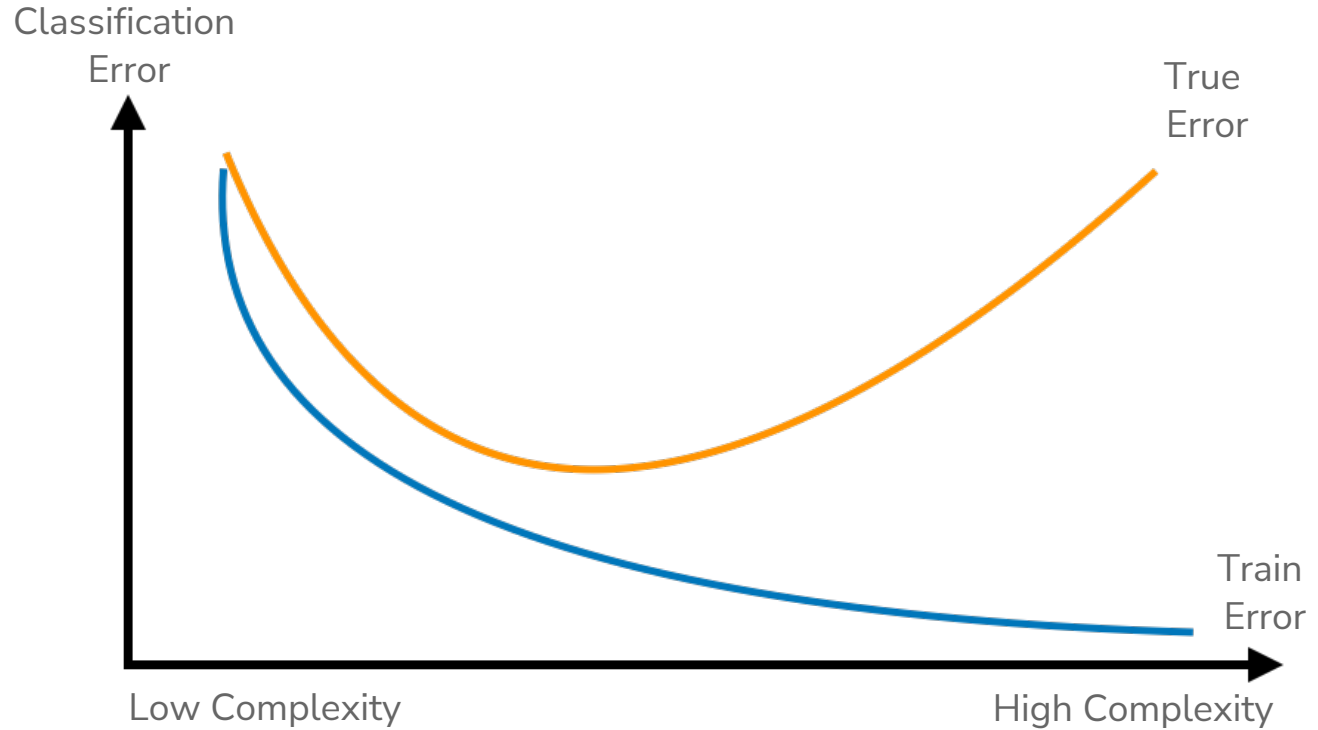
$$w^T h(x) = \dots$$

Feature	Value	Coefficient learned
$h_0(x)$	1	8.7
$h_1(x)$	$x[1]$	5.1
$h_2(x)$	$x[2]$	78.7
...
$h_{11}(x)$	$(x[1])^6$	-7.5
$h_{12}(x)$	$(x[2])^6$	3803
$h_{13}(x)$	$(x[1])^7$	21.1
$h_{14}(x)$	$(x[2])^7$	-2406
...
$h_{37}(x)$	$(x[1])^{19}$	$-2 \cdot 10^{-6}$
$h_{38}(x)$	$(x[2])^{19}$	-0.15
$h_{39}(x)$	$(x[1])^{20}$	$-2 \cdot 10^{-8}$
$h_{40}(x)$	$(x[2])^{20}$	0.03



Overfitting

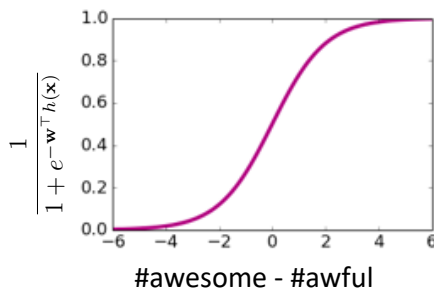
Just like with regression, we see a similar pattern with complexity



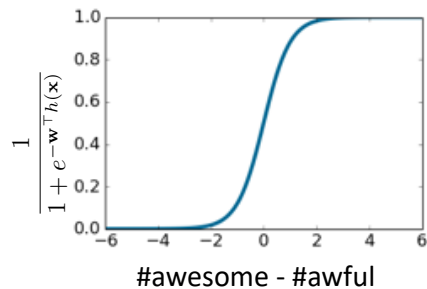
Effects of Overfitting

Remember, we say the logistic function become “sharper” with larger coefficients.

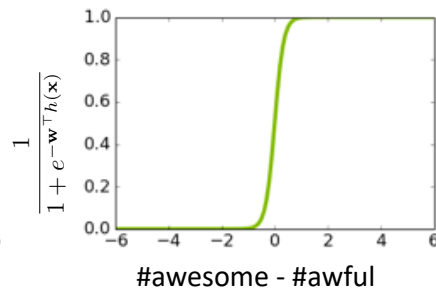
w_0	0
$w_{\text{\#awesome}}$	+1
$w_{\text{\#awful}}$	-1



w_0	0
$w_{\text{\#awesome}}$	+2
$w_{\text{\#awful}}$	-2



w_0	0
$w_{\text{\#awesome}}$	+6
$w_{\text{\#awful}}$	-6

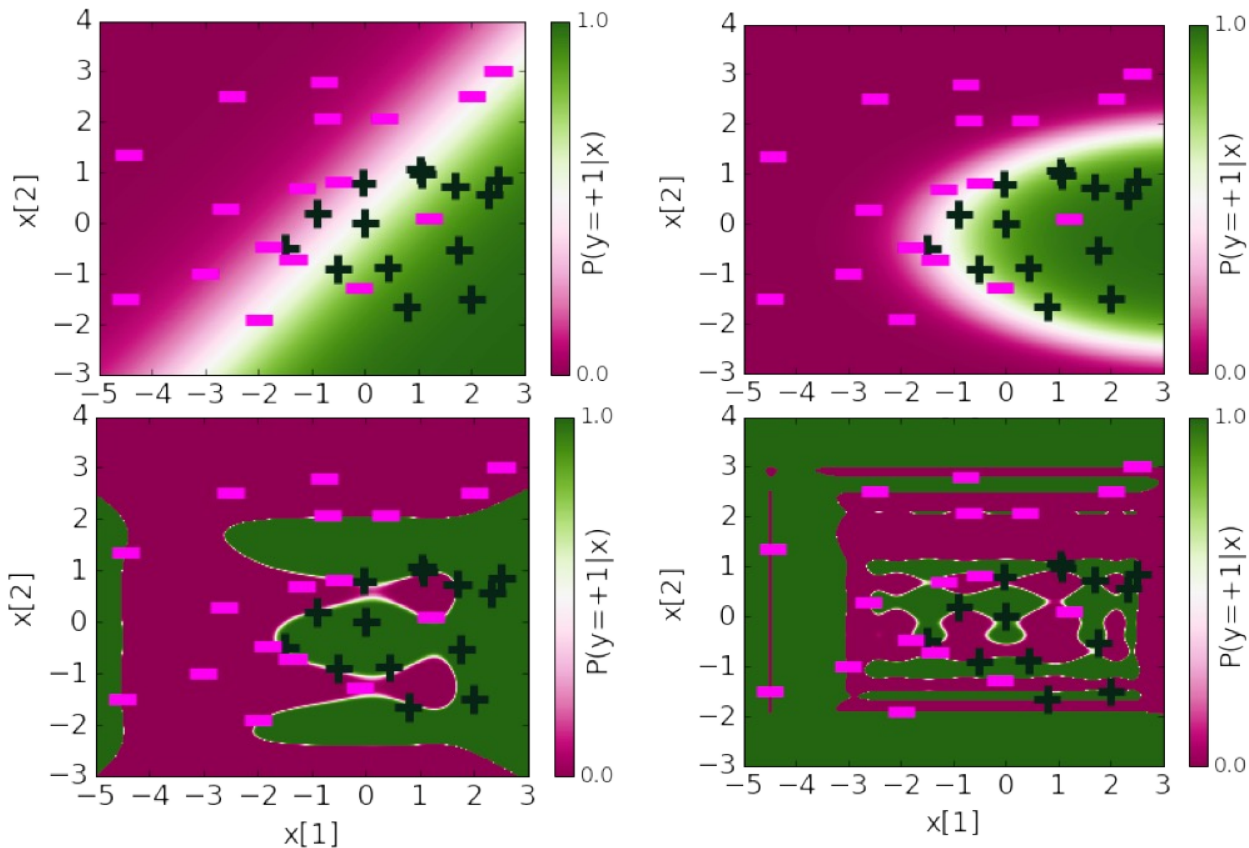


What does this mean for our predictions?

Because the $Score(x)$ is getting larger in magnitude, the probabilities are closer to 0 or 1!

Plotting Probabilities

$$P(y = +1|x) = \frac{1}{1 + e^{-\hat{w}^T h(x)}}$$

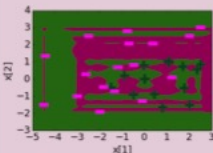
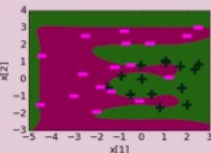
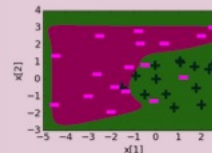
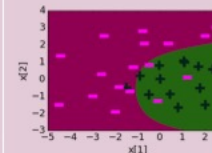
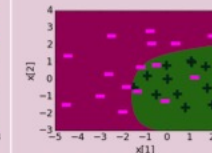
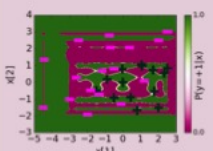
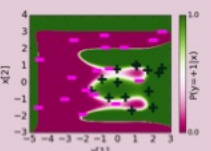
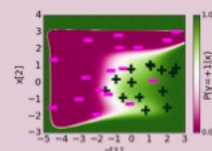
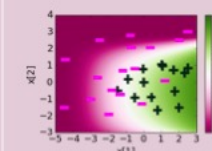



Regularization

L2 Regularized Logistic Regression

Just like in regression, can change our quality metric to avoid overfitting when training a model

$$\hat{w} = \underset{w}{\operatorname{argmin}} L(w) + \lambda \|w\|_2^2$$

Regularization	$\lambda = 0$	$\lambda = 0.00001$	$\lambda = 0.001$	$\lambda = 1$	$\lambda = 10$
Range of coefficients	-3170 to 3803	-8.04 to 12.14	-0.70 to 1.25	-0.13 to 0.57	-0.05 to 0.22
Decision boundary					
Learned probabilities					

Some Details

Why do we subtract the L2 Norm?

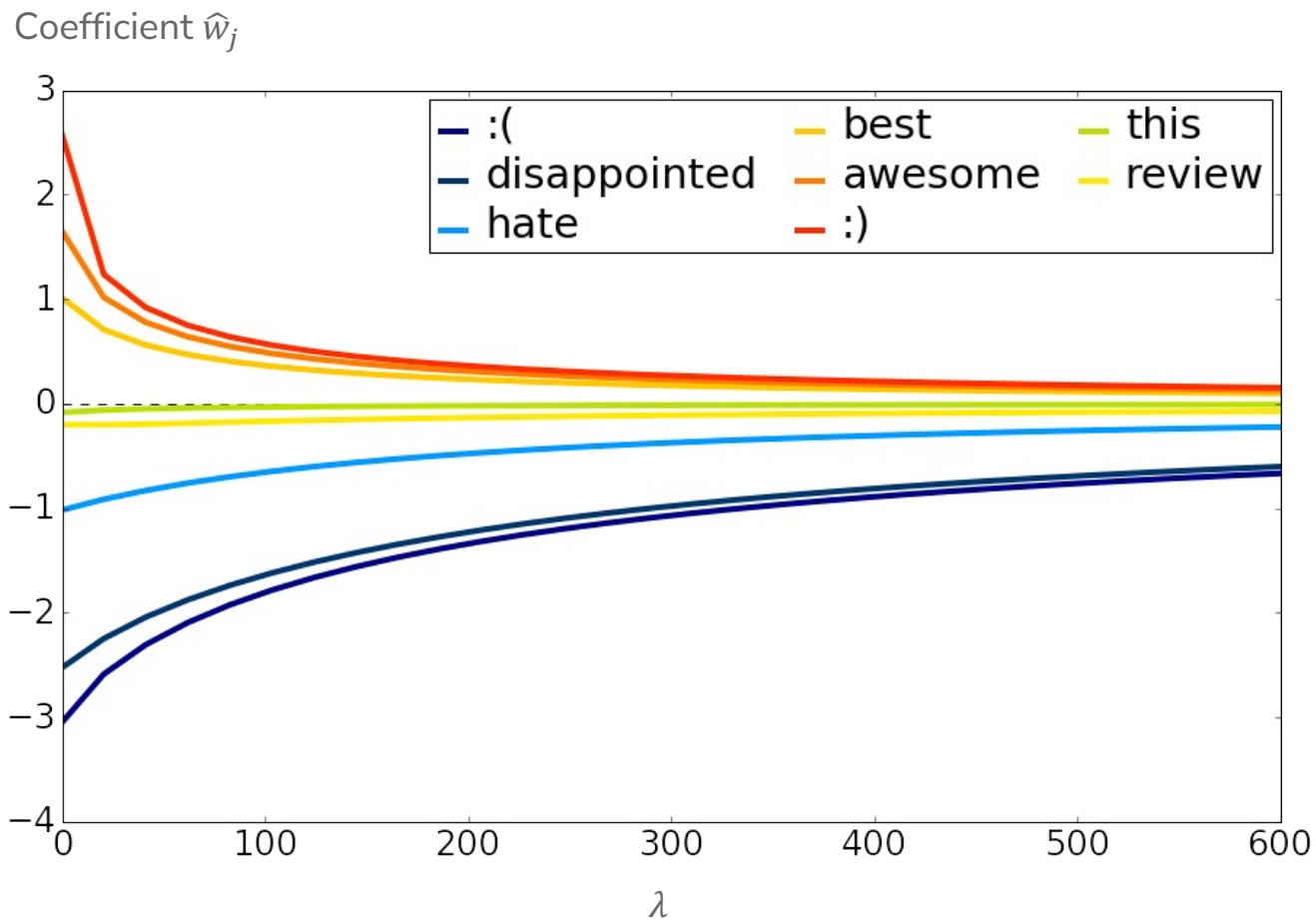
$$\hat{w} = \underset{w}{\operatorname{argmin}} L(w) + \lambda ||w||_2^2$$

How does λ impact the complexity of the model?

How do we pick λ ?



Coefficient Path: L2 Penalty



Other Penalties?

Could you use the L1 penalty instead? Absolutely!

$$\hat{w} = \underset{w}{\operatorname{argmin}} L(w) + \lambda \|w\|_1$$

This is **L1 regularized logistic regression**

It has the same properties as the LASSO

Increasing λ decreases $\|\hat{w}\|_1$

The LASSO favors sparse solutions



Recap

Theme: Details of logistic classification and how to train it

Ideas:

Minimizing error vs maximizing likelihood

Predict with probabilities

Using the logistic function to turn Score to probability

Logistic Regression

Gradient Descent

Effects of learning rate

Overfitting with logistic regression

- Over-confident
- Regularization

